Overture Software for Solving PDEs in Complex Geometry

David L. Brown

Center for Applied Scientific Computing

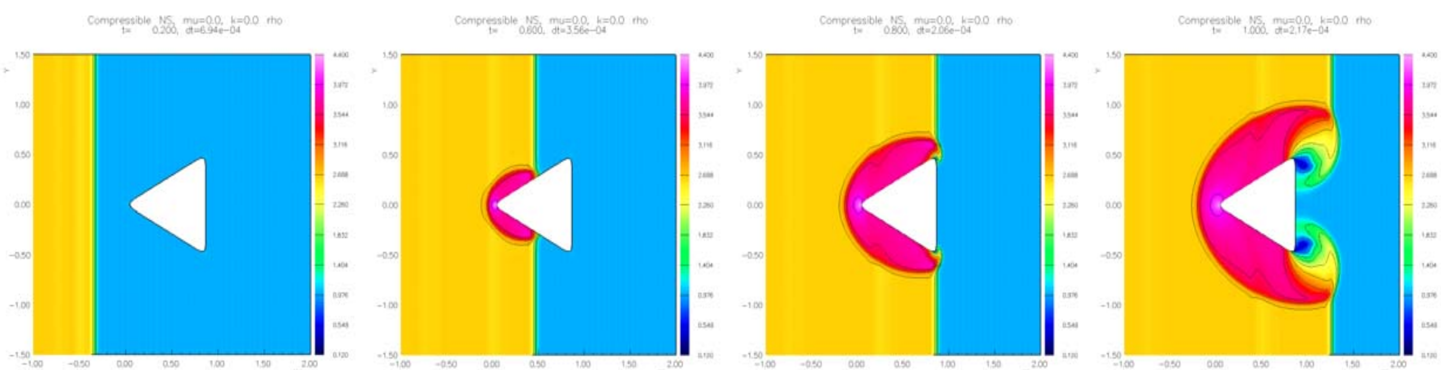
Lawrence Livermore National Laboratory

Presented at ACTS Toolkit Meeting
September 2002

UCRL-PRES-150012

Overture develops algorithms and software for PDEs in complex moving geometry

- **We develop and analyze new algorithms for PDEs in complex moving geometry**
- **The Overture software framework is a flexible test-bed for prototyping new application codes**
- **We have significant recognition in the community for our math and CS-based framework design research**
- **Funded by SC/OASCR and LLNL/LDRD**



Simulation of shock incident on triangular obstruction using Overture

Capabilities provided by Overture v.19 Libraries

- Grid Generation
  - Basic geometry creation tools
  - CAD IGES file clean-up and repair tools *NEW*
  - Structured mapped grid creation from CAD
  - Overset grid generation
  - Hybrid (multi-element) grid generation *NEW*
  - Embedded Boundary (EB) grid generation *NEW*
- PDE Discretization and Solver Building Tools
  - P++ parallel array language
  - Discretization Operators *Now Optimized*
  - Linear solvers *Now with Multigrid*
  - Adaptive Mesh Refinement (AMR) *NEW*

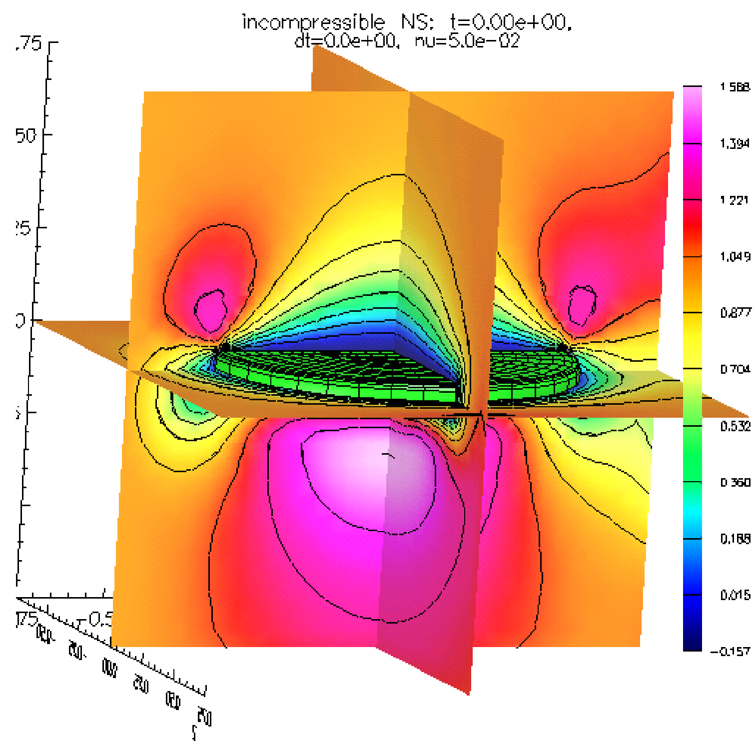
Capabilities provided by Overture v.19 Libraries

- *OverBlown* Flow Solver
  - Incompressible
  - Slightly compressible
  - Compressible
  - *Now with AMR*
- Overture Visualization Tools
- Documentation
  - http://www.llnl.gov/casc/Overture/
  - dlb@llnl.gov for copies of this presentation

Multi-scale simulations in complex moving geometry present many challenges

- **Need to generate high-fidelity representation of complex geometry**
- **Regeneration of grids as geometry evolves**
- **Special algorithms required to address**
  - **flow-driven motion**
  - **free surfaces**
  - **high-order methods**
  - **optimized linear algebra**
  - **multiple time&space scales**

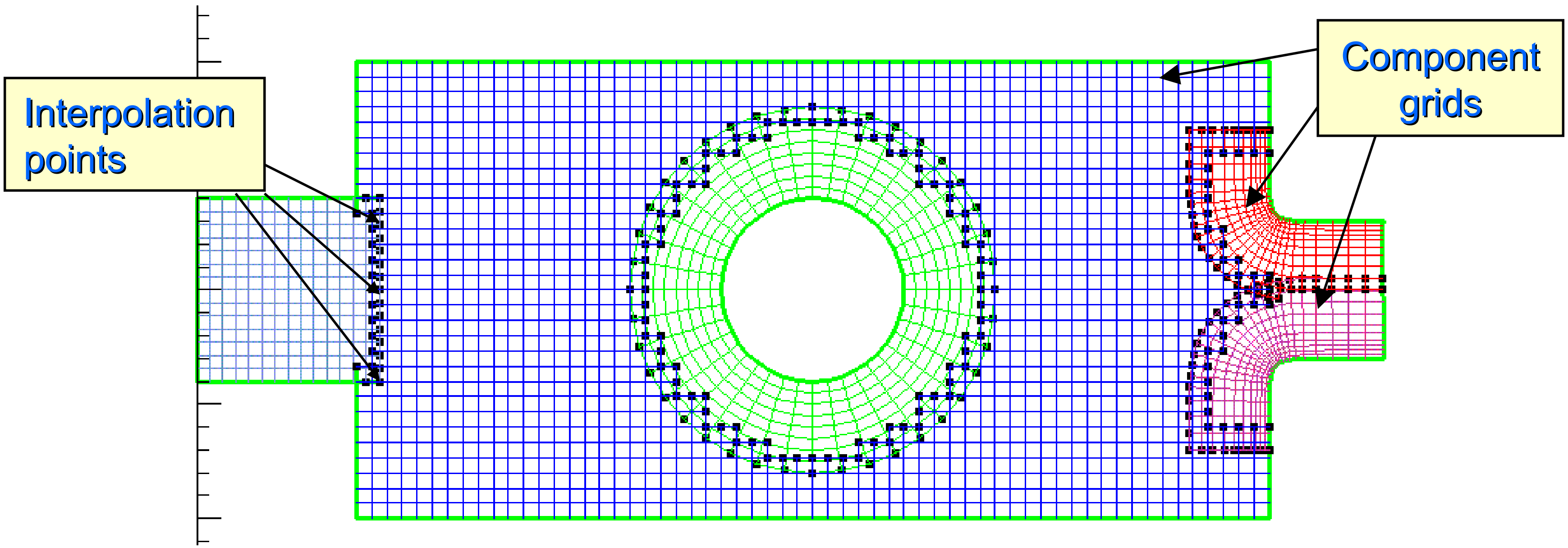
Rotating disc in a viscous fluid simulated using Overture



For complex geometry we use *overlapping grids*

- **Set of logically rectangular curvilinear grids**
- **Overlap where they meet**
- **Completely cover the domain**

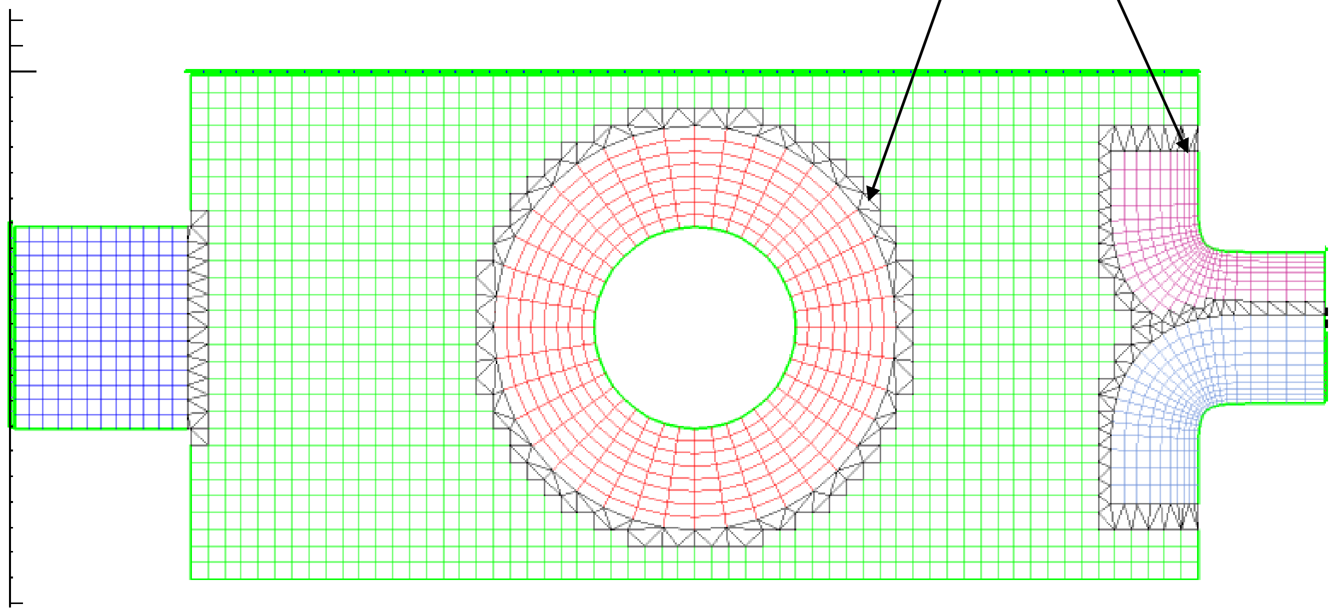
2D grid for pipe with inlet and outlet ports and a cylindrical obstruction



The Overture grid generator, Ogen, automatically computes the connectivity information.

For complex geometry we use *overlapping grids* or *mixed-element grids*

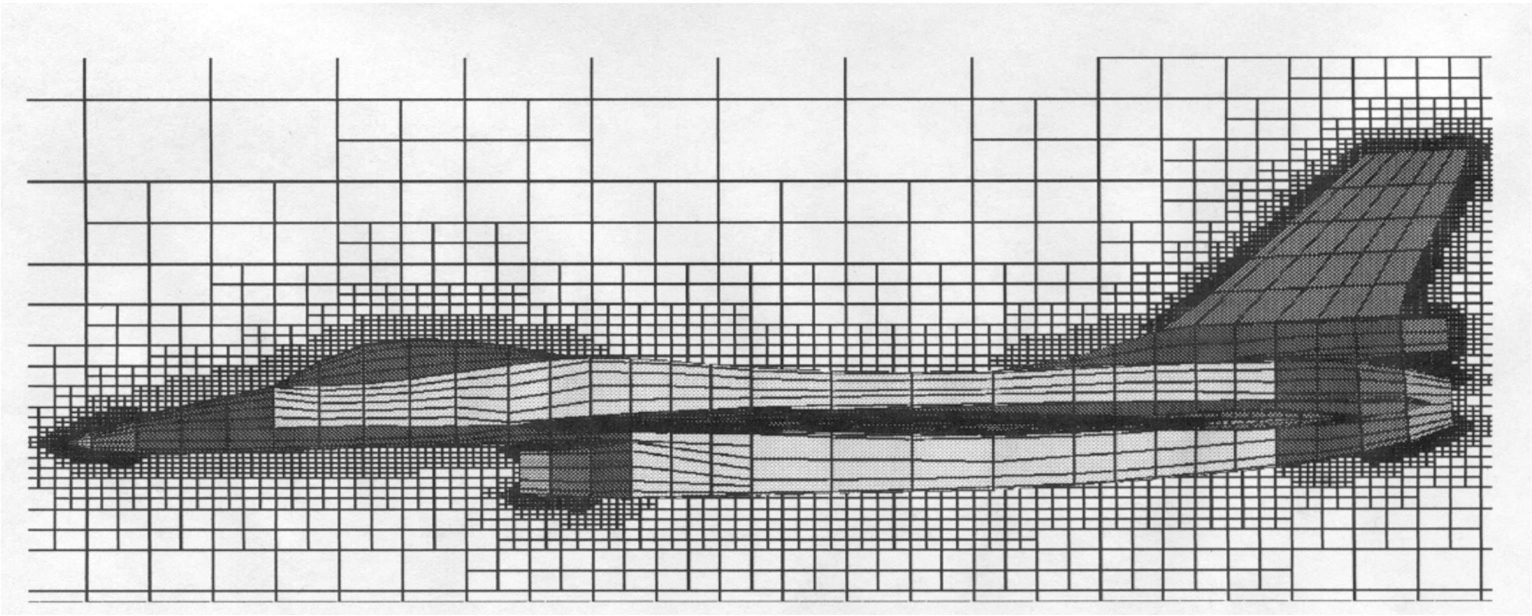
- Set of logically rectangular curvilinear grids
- Overlap where they meet
- Completely cover the domain



Unstructured connections

Mixed-element grids make our mesh generation technology available to more application codes.

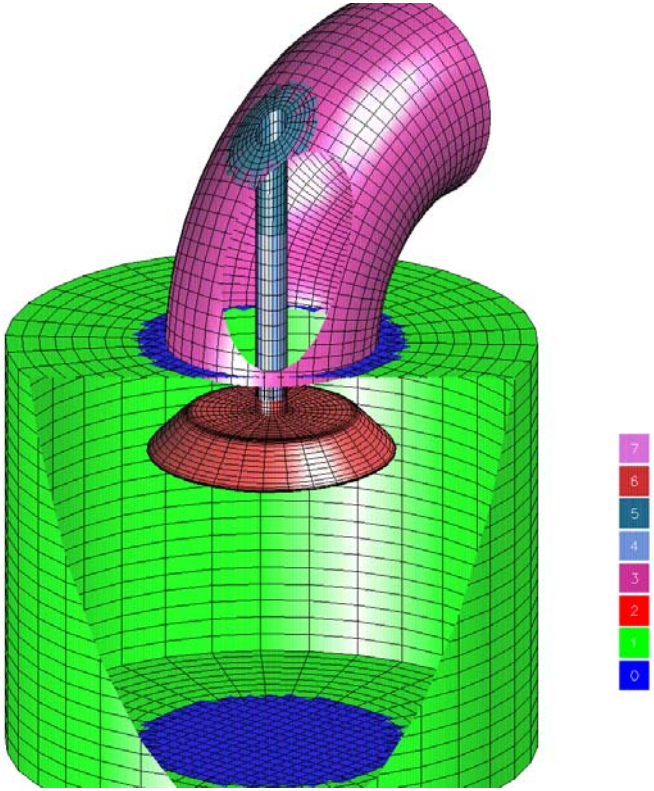
Using our CAD geometry tools we can also build embedded boundary grids



Grid courtesy
Marsha Berger,
CIMS

Overlapping grids have significant strengths for moving geometry

High-fidelity representation of boundary physics

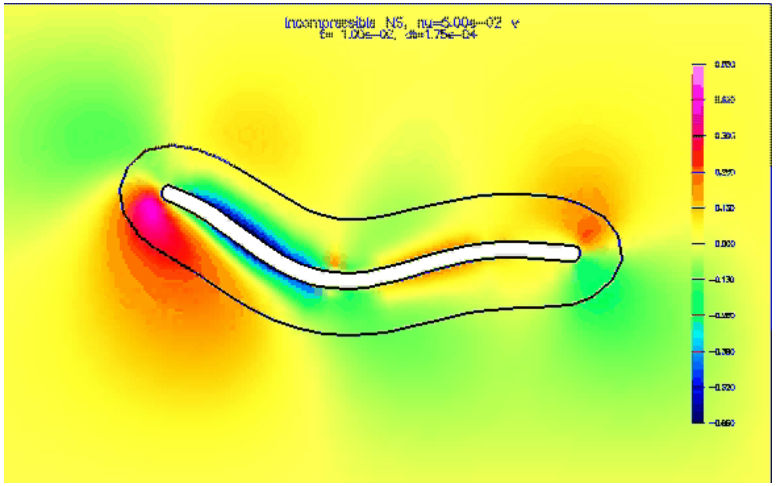


3D grid for cylinder and intake valve

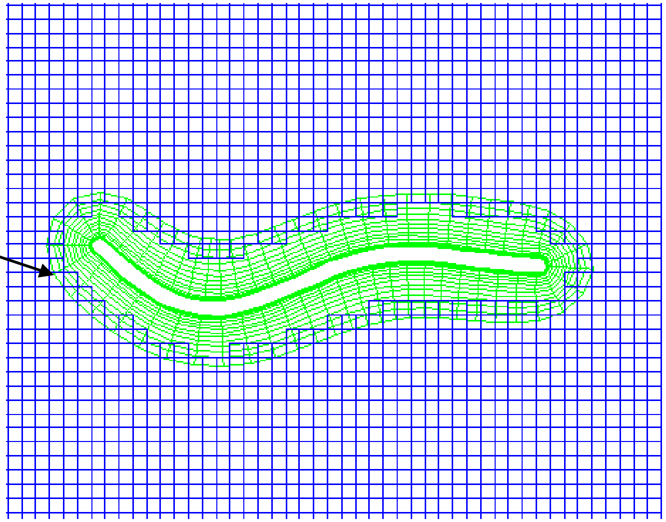
Overlapping grids have significant strengths for moving geometry

High-fidelity representation of boundary physics

Moving geometry is implemented efficiently



Entire mesh does not need to be regenerated when the component moves

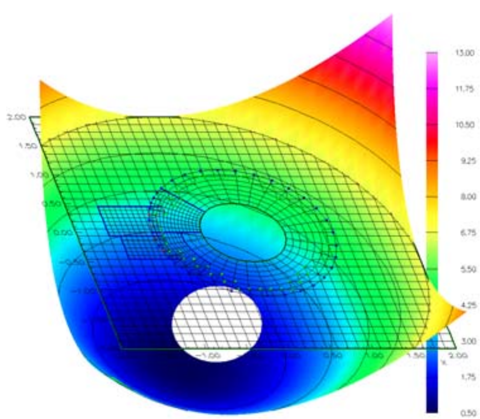


Overlapping grids have significant strengths for moving geometry

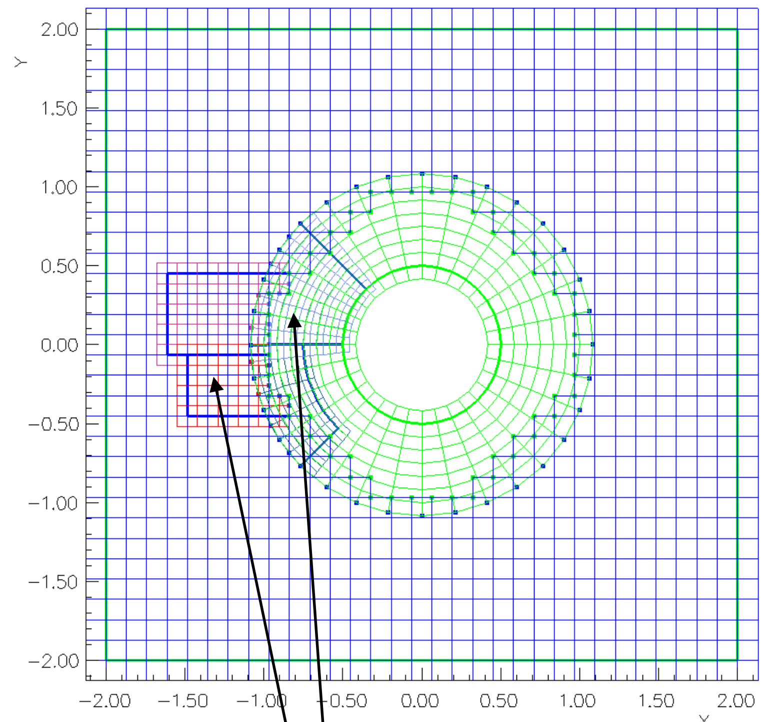
High-fidelity representation of boundary physics

Moving geometry is implemented efficiently

Structured AMR is used



Adaptive Mesh Refinement



Each component grid refined independently

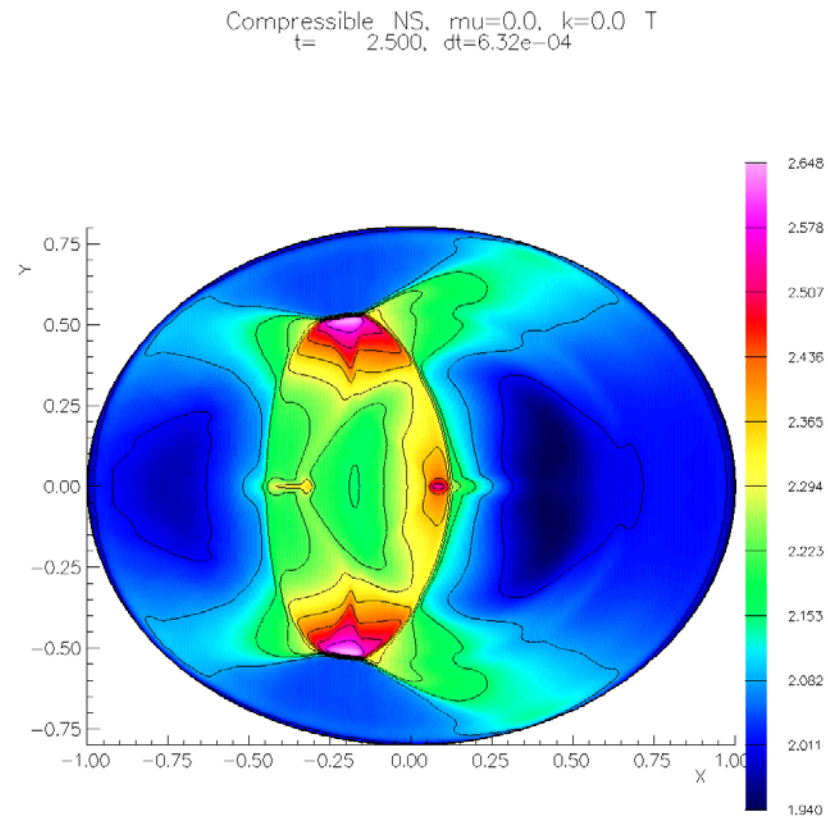
Overlapping grids have significant strengths for moving geometry

High-fidelity representation of boundary physics

Moving geometry is implemented efficiently

Structured AMR is used

Leverage accurate, efficient structured grid algorithms



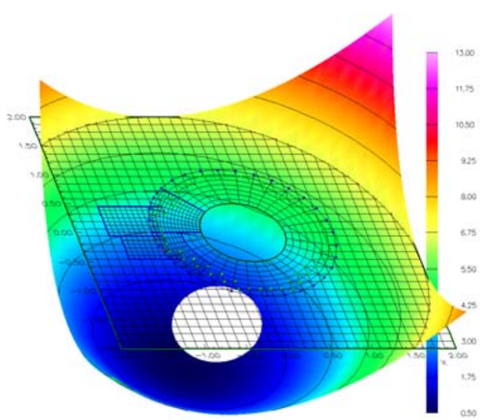
RPI collaboration:
High-order Godunov method applied to reacting flow detonation computation

Overlapping grids have significant strengths for moving geometry

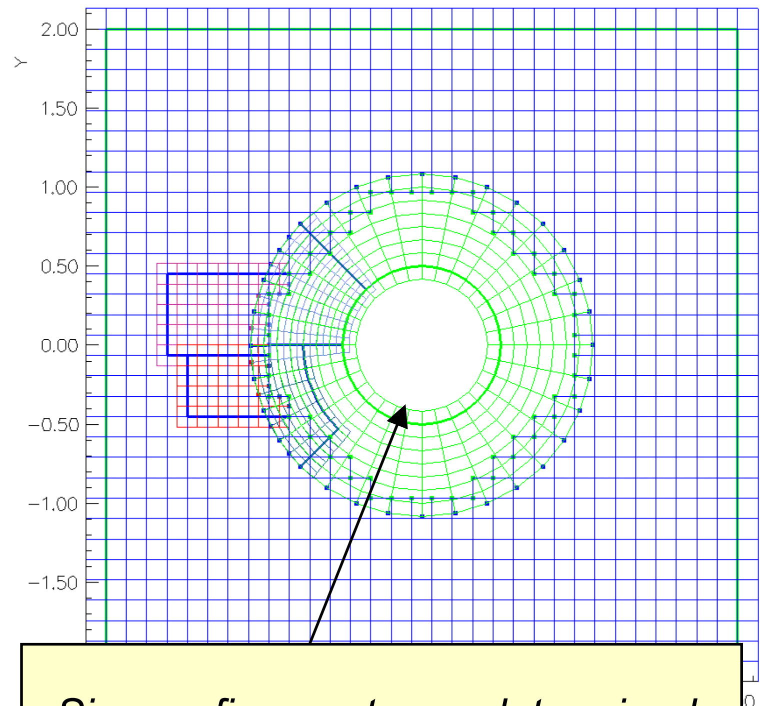
High-fidelity representation of boundary physics

Moving geometry is implemented efficiently

Structured AMR is used



Adaptive Mesh Refinement

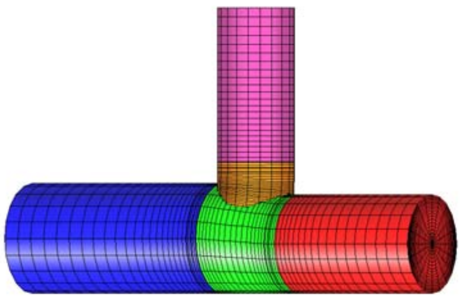


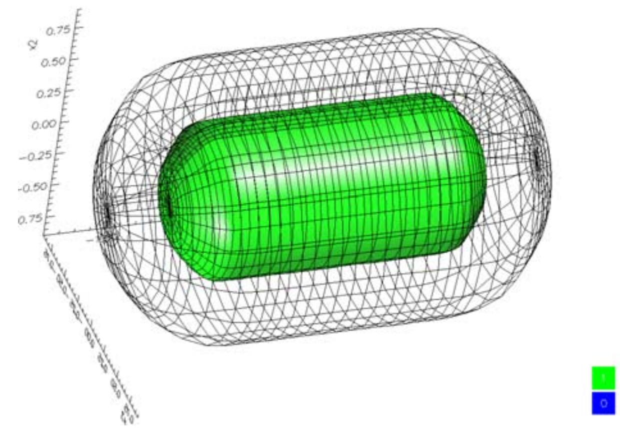
Since refinements are determined dynamically, access to original geometry required at run-time

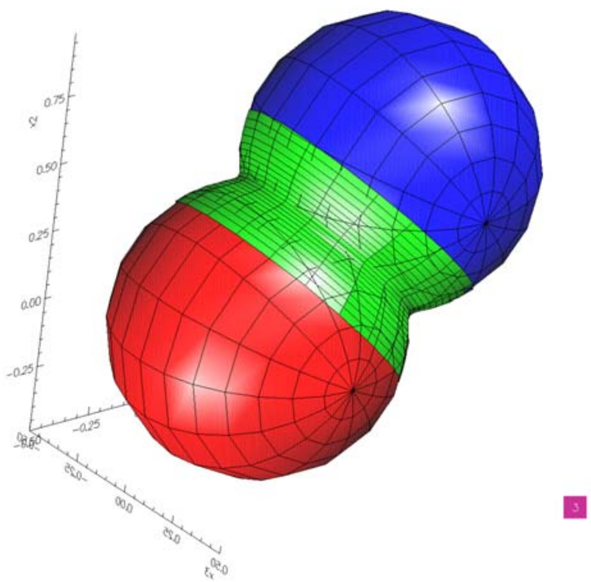
Overture Grid Generation Tools “Rapsodi” (rapid setup)

- Ogen:
  - Build mapped grids (actually mappings)
  - Build overset grids
- Ugen:
  - Build unstructured grids
  - Build hybrid grids
- Rap:
  - Build mapped grids from CAD
  - Repair and Modify CAD files
  - Build triangulated watertight surfaces for input in CART3D (EB grids)

Ogen can be used to create mappings for simple geometries



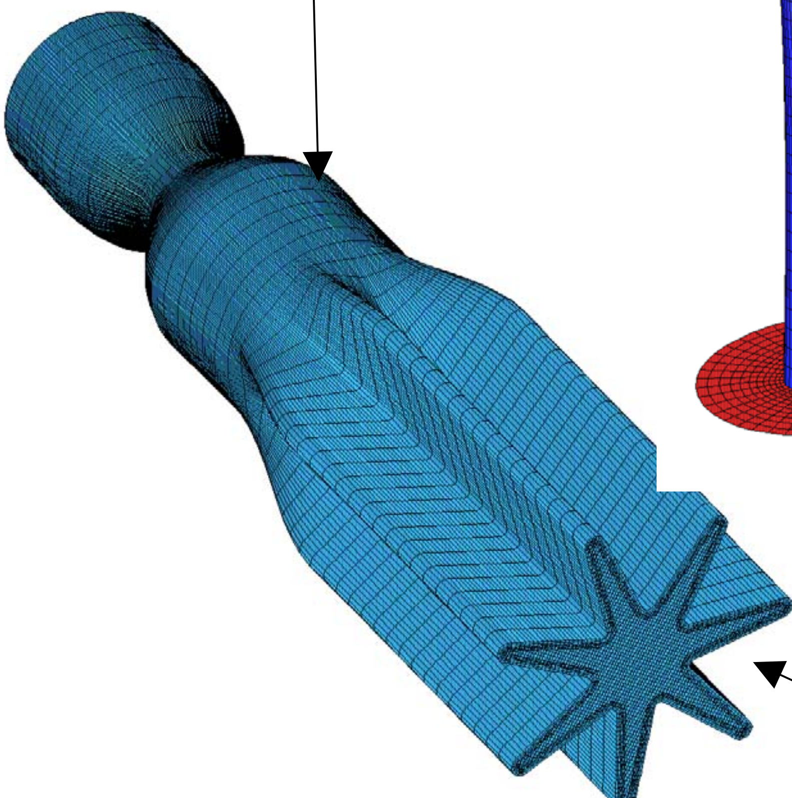


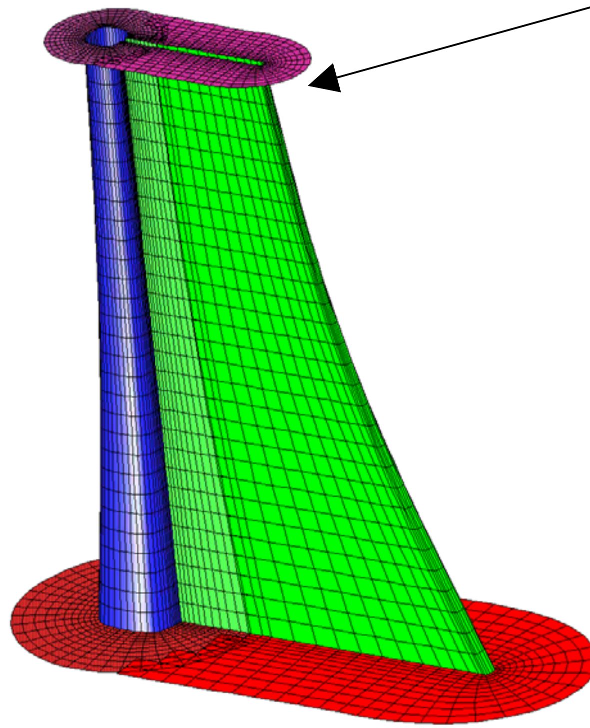


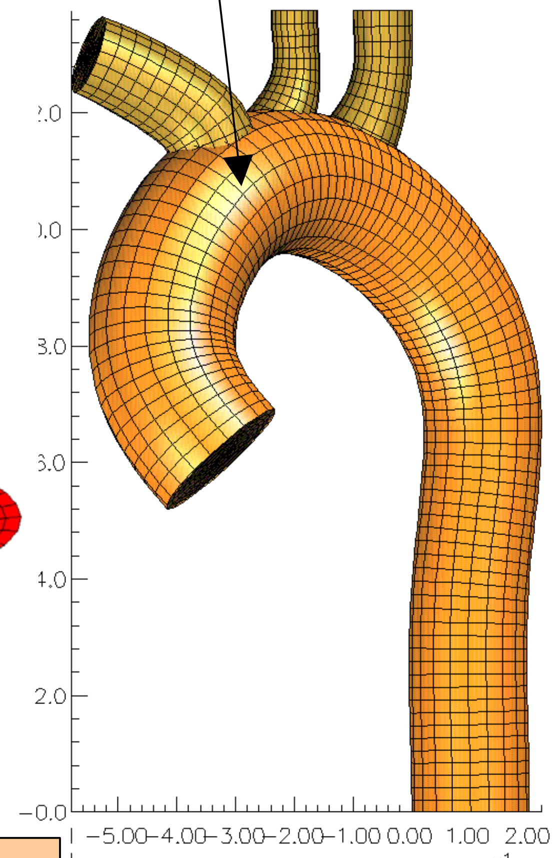
More mappings from simple geometries

Sweep mapping

Cross-section mapping

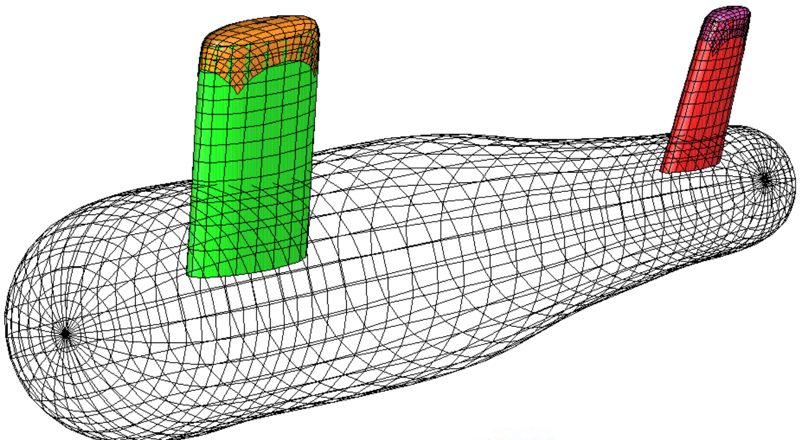




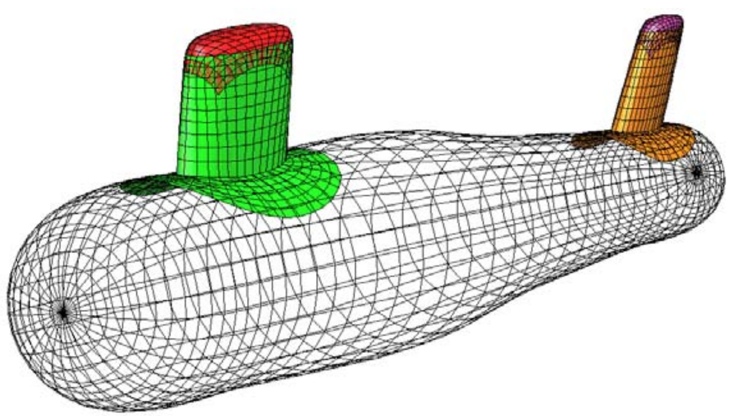


Rocket cross-sections

The overlapping approach is based on component assembly

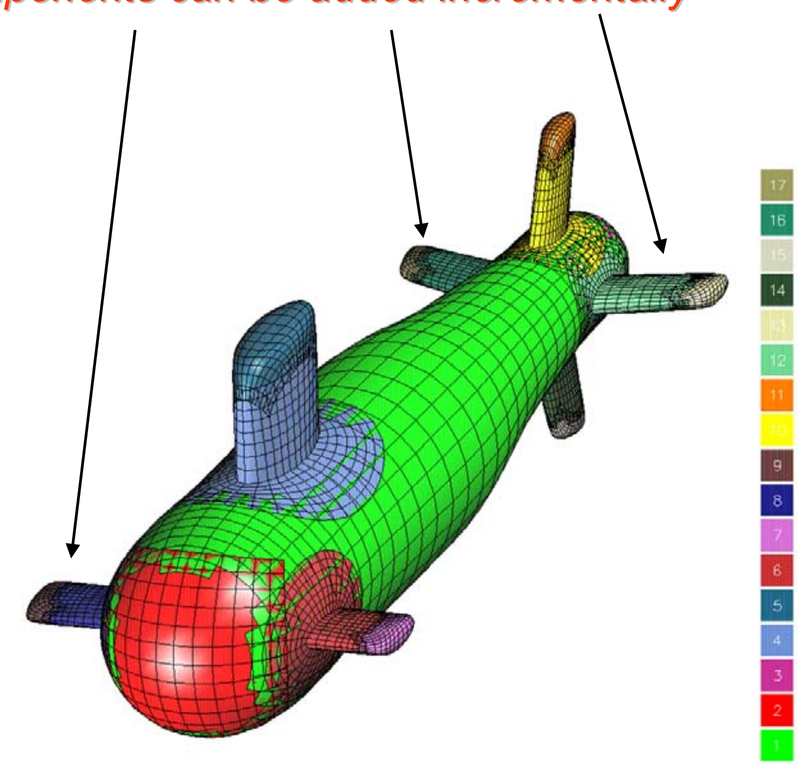


1. *Components assembled*



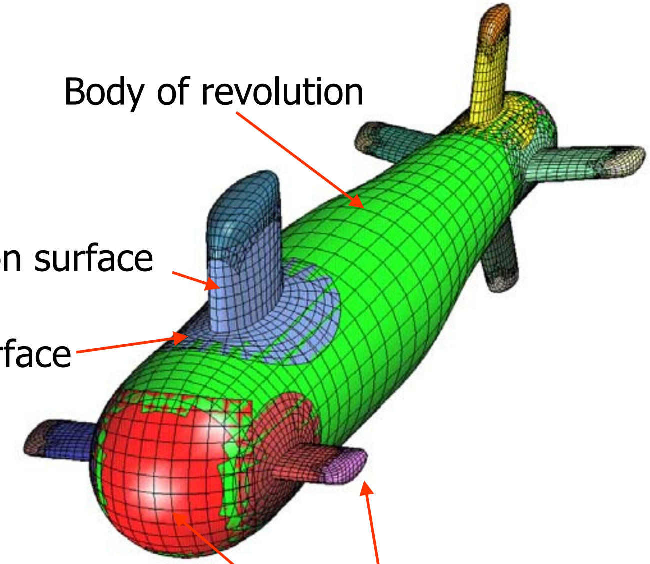
2. *Intersections computed automatically; blended to submarine body surface*

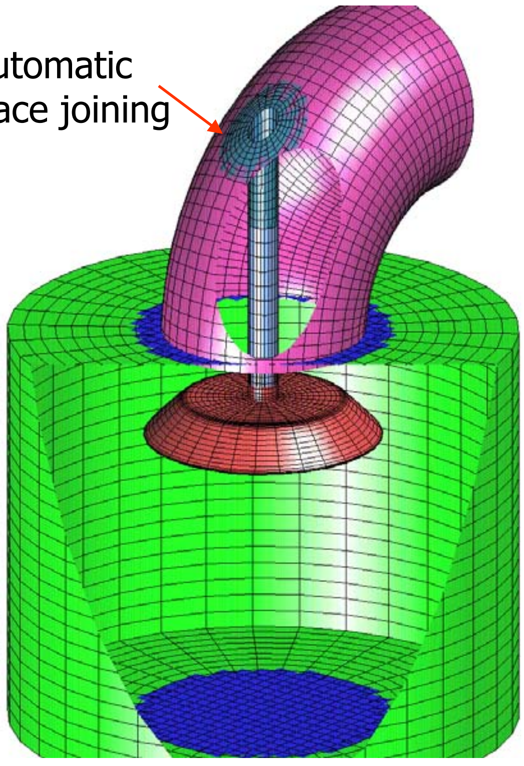
Components can be added incrementally



3. *Final overset grid*

Overset Grid Generation Capabilities



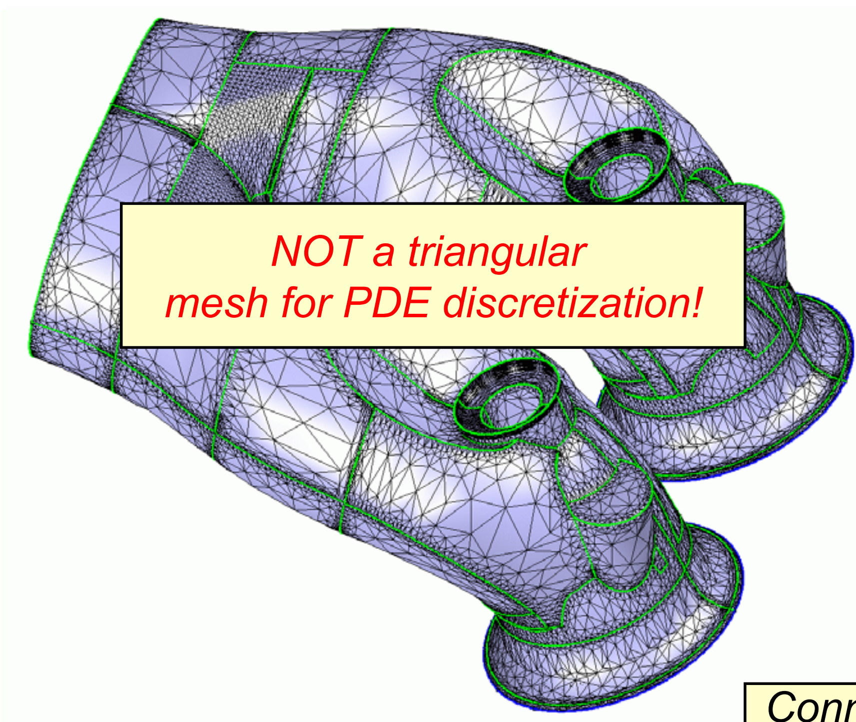


Bill Henshaw

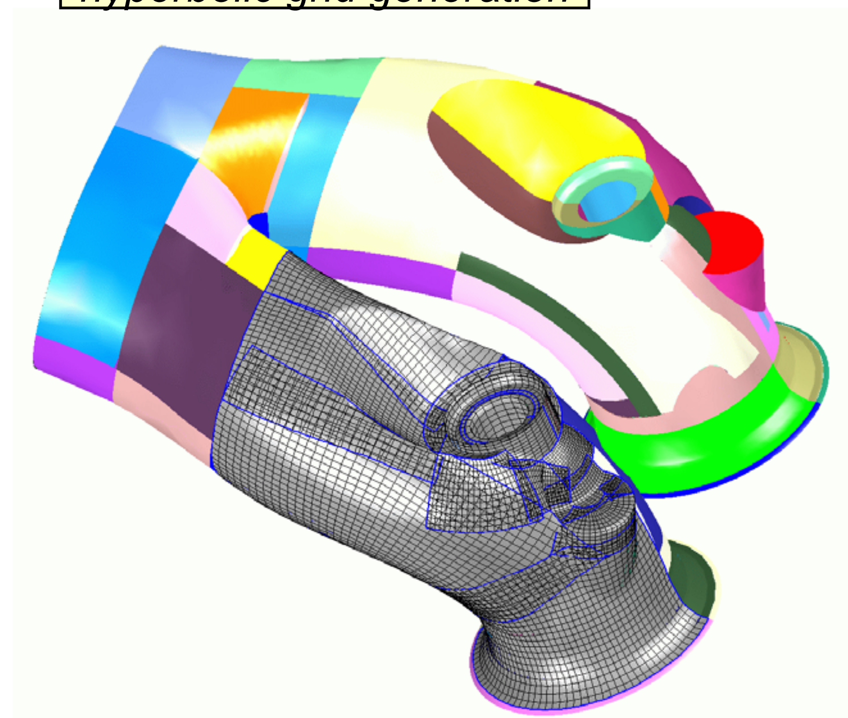
We have developed tools to build component grids on CAD models

Original (unrepaired) CAD

Surface grids by hyperbolic grid generation



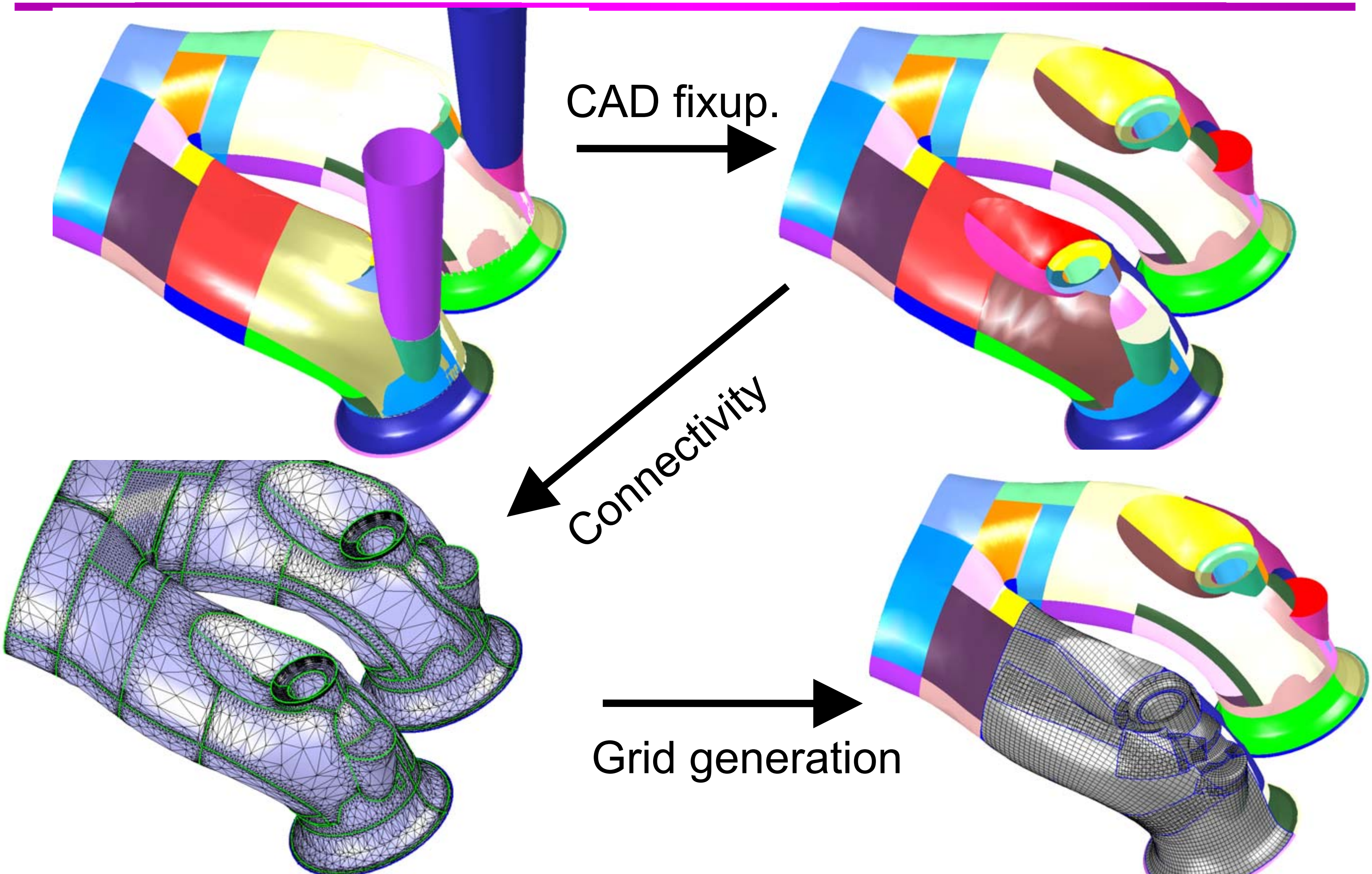
NOT a triangular mesh for PDE discretization!



CAD is repaired using Rap

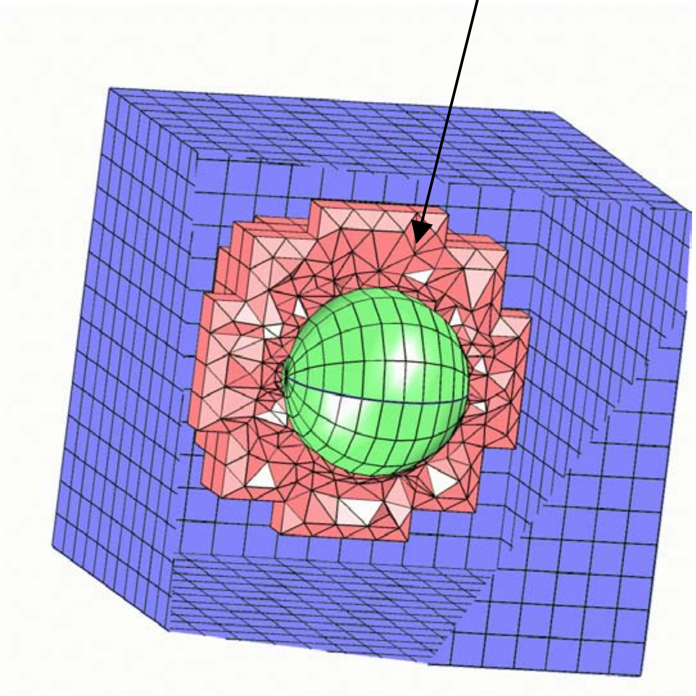
Connectivity of patches determined

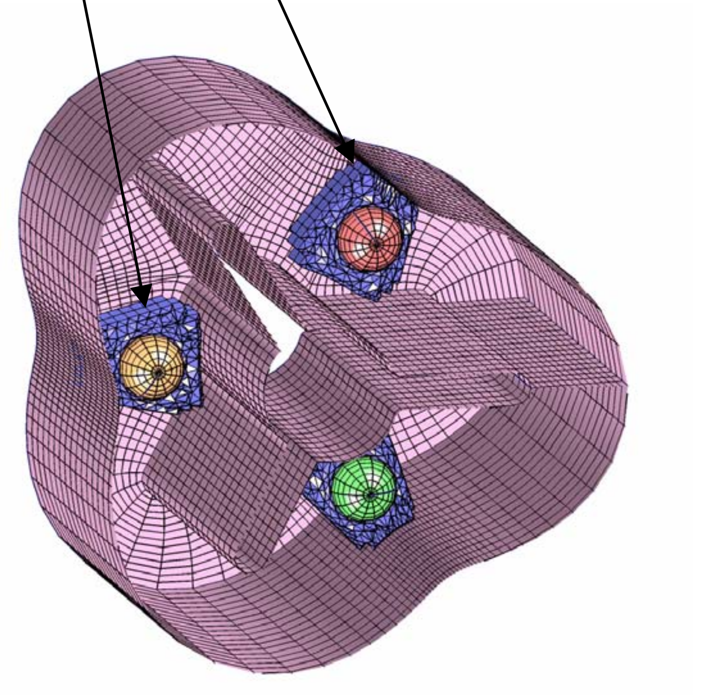
We have developed tools to build component grids on CAD models



We can also build 3D mixed-element meshes

Unstructured connection meshes are built with tetrahedra and pyramids

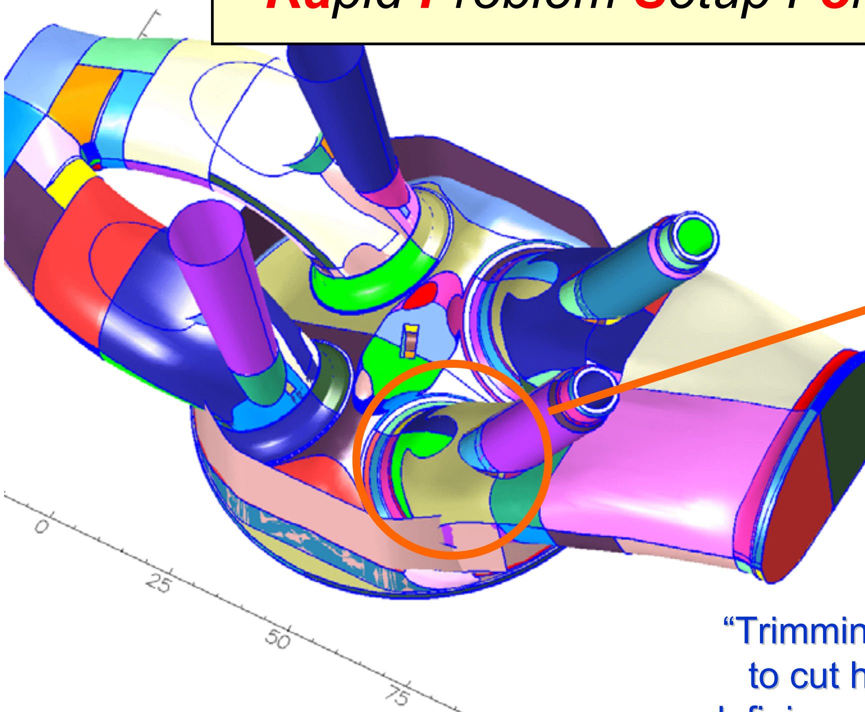


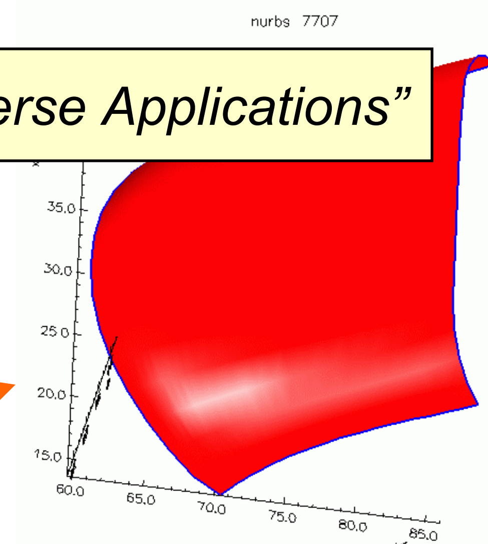


Rap has automated tools for pre-grid-generation CAD geometry repair

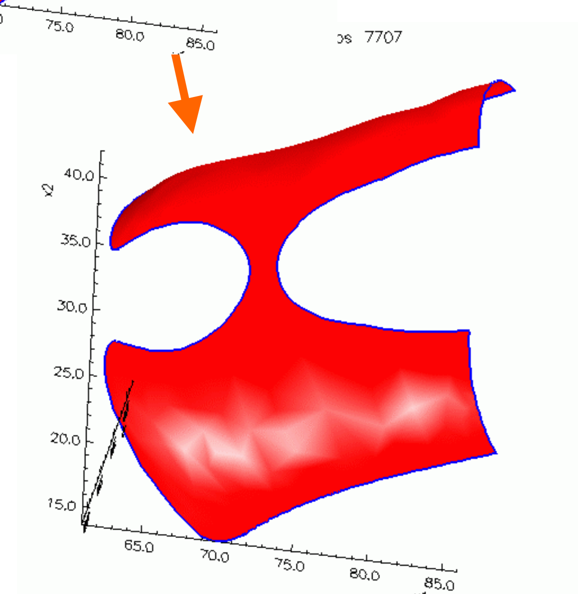
Volvo intake manifold

*“**Ra**pid **P**roblem **S**etup F**o**r **Di**verse Applications”*





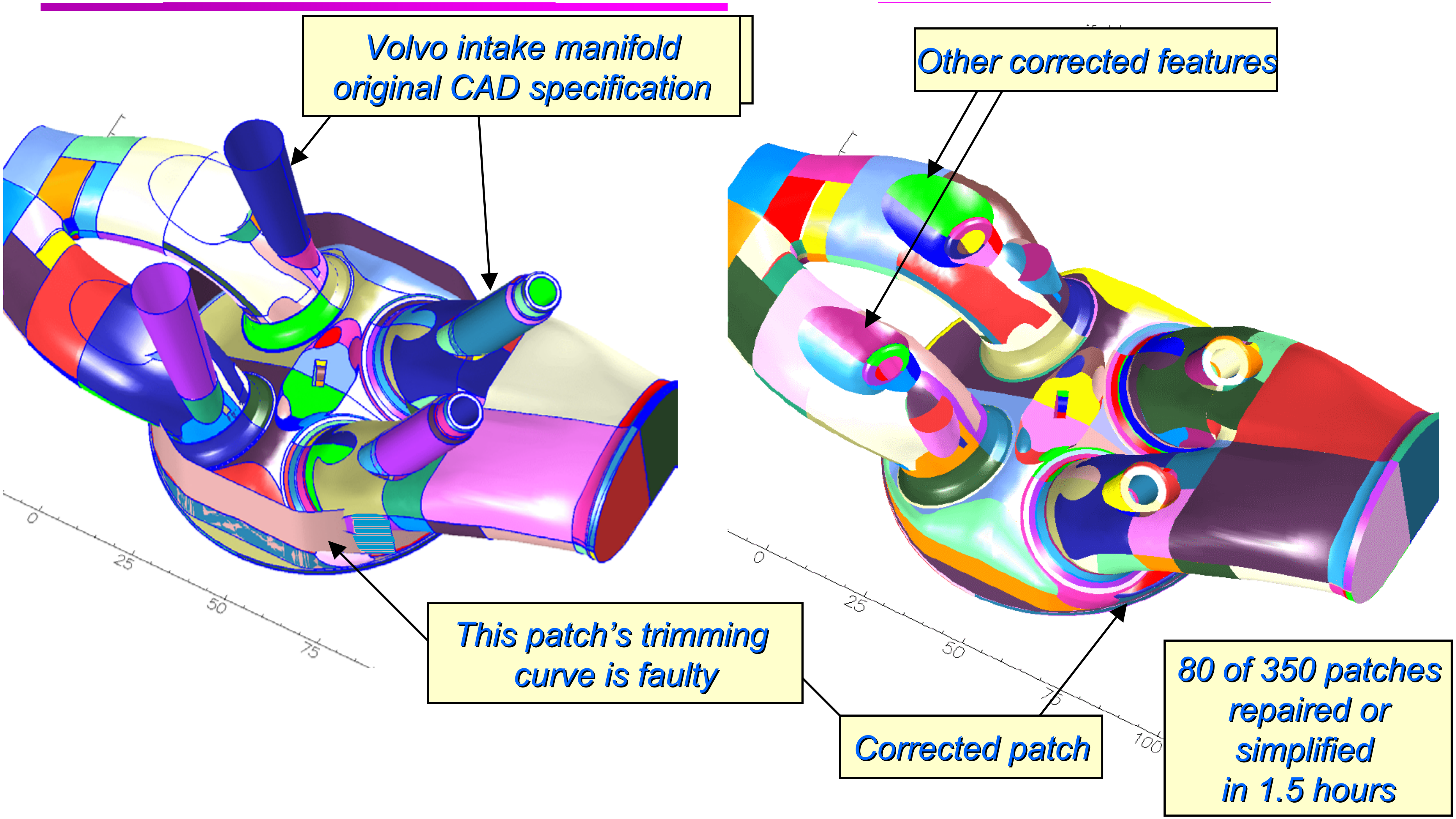
Each geometry element is represented by a logically rectangular spline patch (“NURBS”)



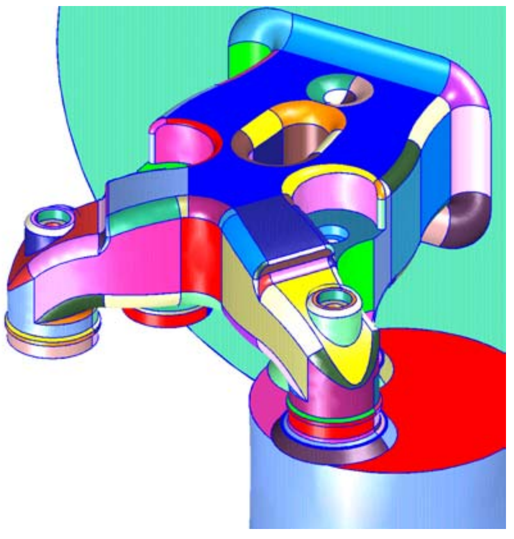
“Trimming curves” are used to cut holes in the patch, defining more general shapes

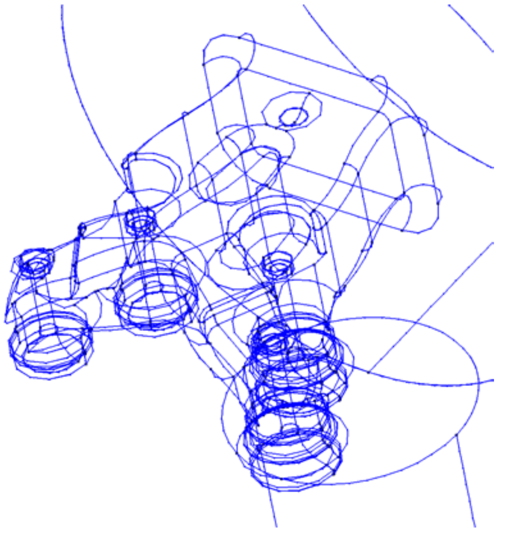
We automatically detect trimming curve errors in CAD

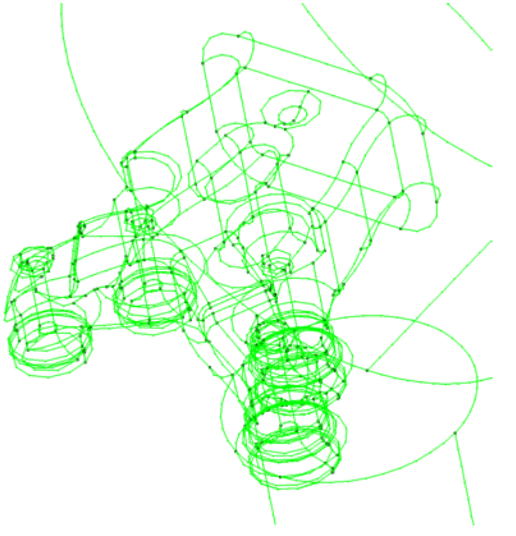
Rap has automated tools for pre-grid-generation CAD geometry repair



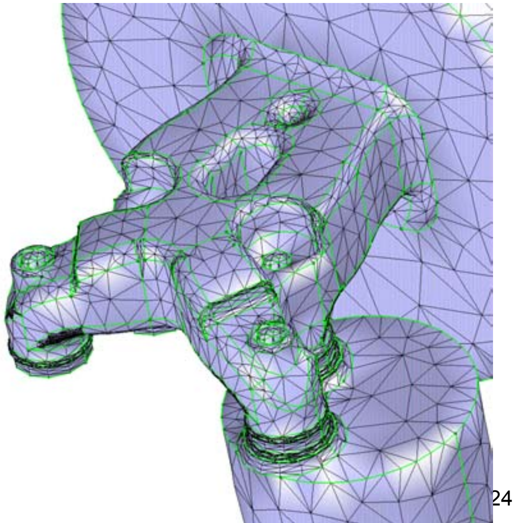
The connectivity of the CAD model is determined automatically



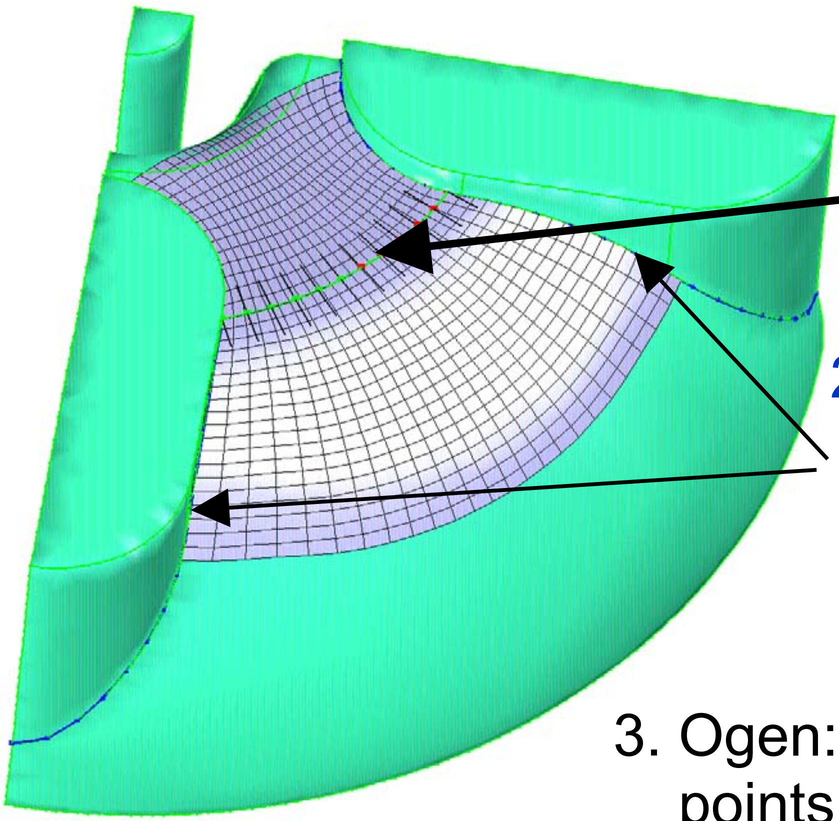




1. CAD model from IGES file (no connectivity)
2. Build edge curves of all trimmed surfaces
3. Match edges where surfaces meet.
4. Triangulate patches and stitch together



Mappings are built on CAD surfaces using hyperbolic grid generation

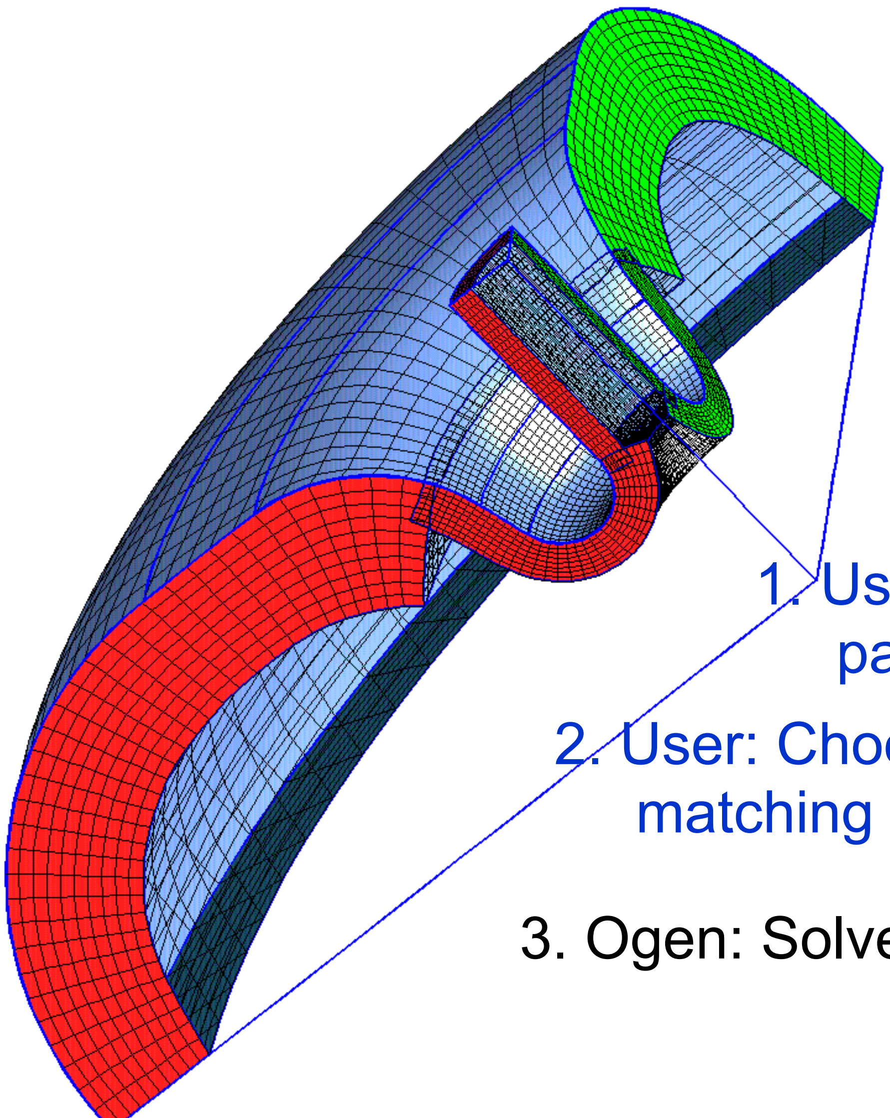


1. User chooses initial curve.

2. User chooses boundary conditions, (matching boundaries are automatically found) and marching parameters.

3. Ogen: Grid built by marching. At each step points projected onto the CAD surface using the global triangulation as an initial guess.

Hyperbolic volume grid generation is a similar process

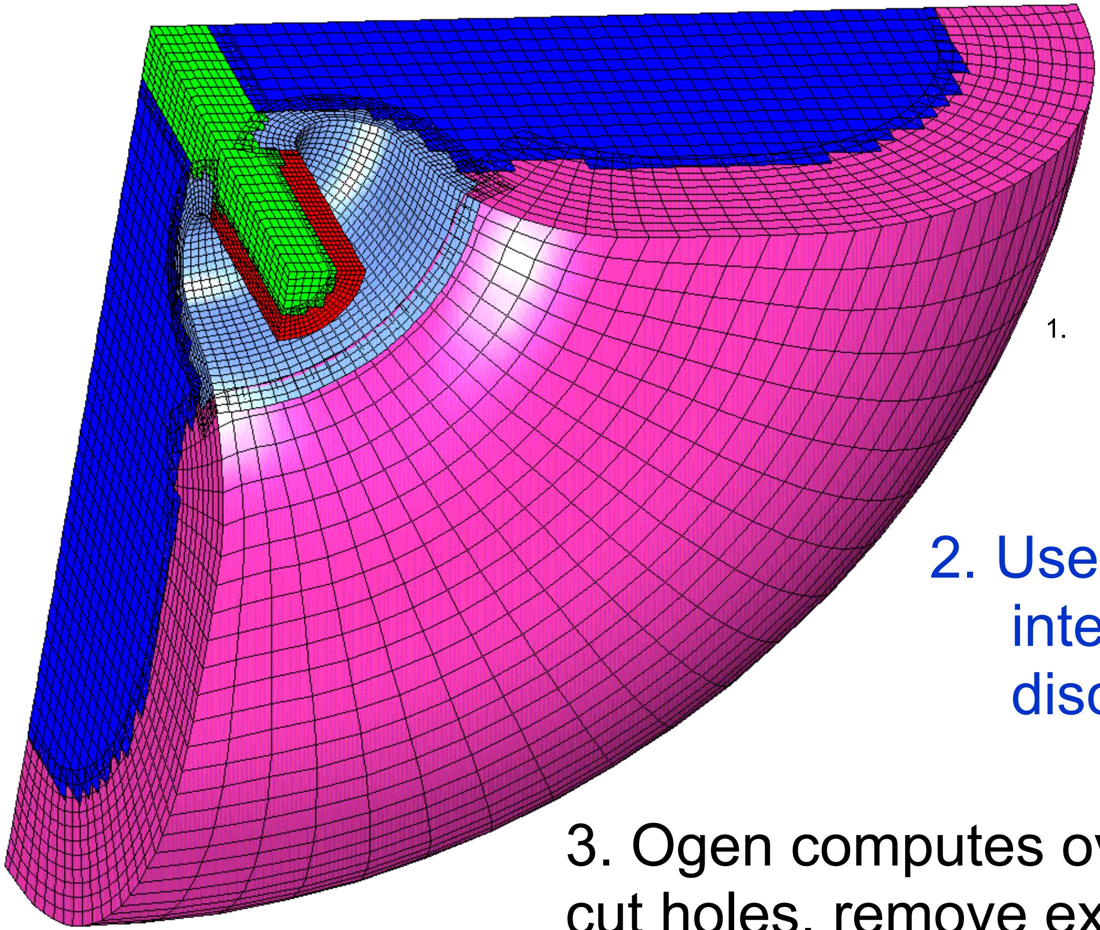


1. User: Choose marching distance and parameters.

2. User: Choose boundary conditions (such as matching to an adjacent surface).

3. Ogen: Solve hyperbolic marching equations.

Generation of the overlapping grid is also an automatic process



1. 1. User: Specify physical, interpolation and "shared" boundaries.

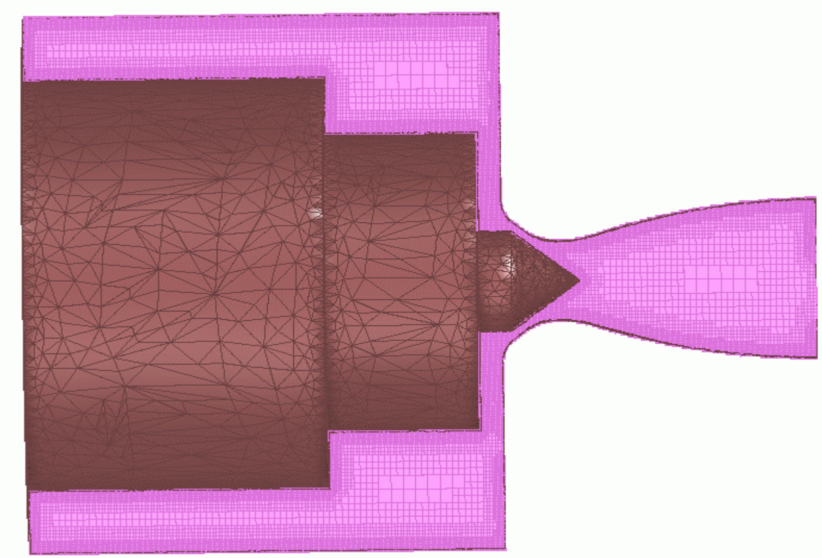
2. User: Choose overlap parameters, interpolation width, discretization width, ...

3. Ogen computes overlap: cut holes, remove excess overlap.

APDEC brings AMR and embedded boundary mesh technology to SciDAC apps

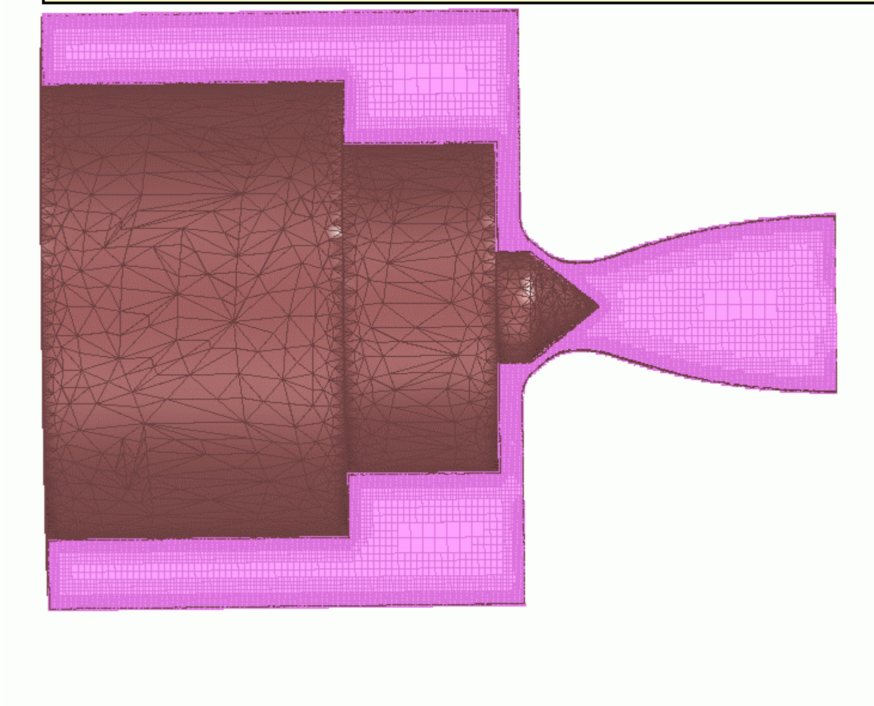
- **AMR MHD Tokomak simulation code (with Princeton Plasma Physics Lab)**
- **Electrostatic Particle-Mesh code for accelerator simulation**
- **Direct numerical simulation of combustion**
- **Simulation of compressible jets for laser-plasma accelerators**

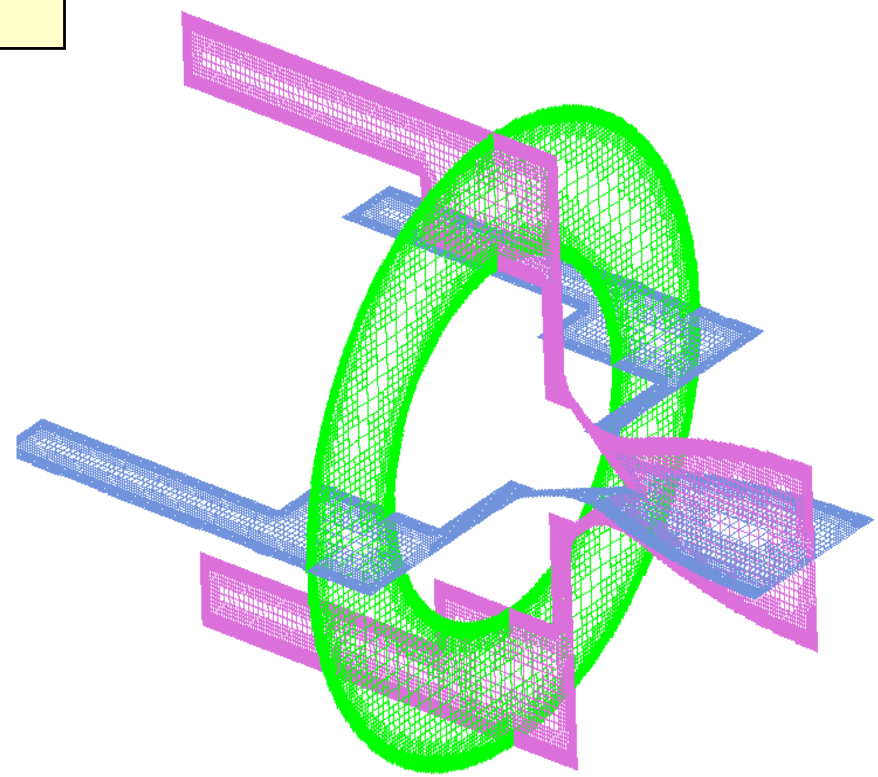
Phil Colella,PI



Rapsodi is developing CAD-to-mesh tools for APDEC

Rapsodi simple CAD functionality used to build this geometry





Embedded boundary grid for SciDAC/APDEC nozzle geometry uses Rapsodi tools and CART3D (NASA/NYU)

Overture Discretization Tools

- P++ parallel array language
- Discretization Operators
- Linear Solvers
- Adaptive Mesh Refinement
- Rapid prototyping of PDE applications
- Visualization

The fundamental building block for the *Overture* framework is the P++ array class

Stencil operations on structured grids are naturally expressed in terms of array operations

Details of parallel implementation can be hidden from the user by the array class

Like F90 Index Triplet

Parallel communication occurs at the =

Like F90 Arrays

```
Index I,J;
floatArray u,v,w;

// update stencil and communicate between processors

u(I,J) +=  .25*(u(I-1,J) + u(I+1,J) + u(I,J-1) + u(I,J+1));
```

The P++ Array class is the basis for all higher-level objects and functions in *Overture*

P++ provides array objects and operations in a parallel environment

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

P++ *Arrays* are distributed over many processors on a parallel machine

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

P++ *Arrays* are distributed over many processors on a parallel machine

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

P++ *Arrays* are distributed over many processors on a parallel machine

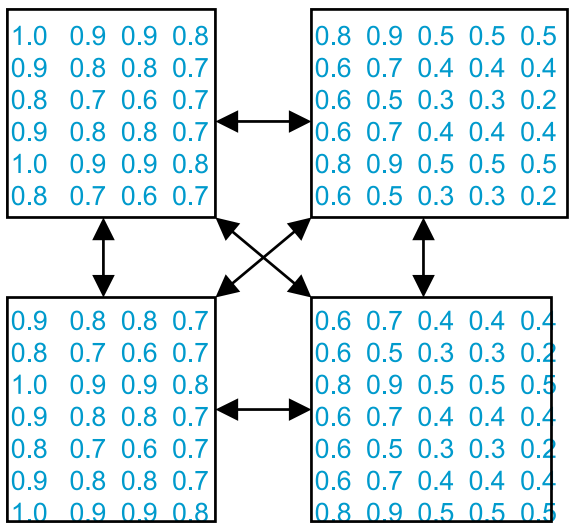
1.0	0.9	0.9	0.8
0.9	0.8	0.8	0.7
0.8	0.7	0.6	0.7
0.9	0.8	0.8	0.7
1.0	0.9	0.9	0.8
0.8	0.7	0.6	0.7

0.8	0.9	0.5	0.5	0.5
0.6	0.7	0.4	0.4	0.4
0.6	0.5	0.3	0.3	0.2
0.6	0.7	0.4	0.4	0.4
0.8	0.9	0.5	0.5	0.5
0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7
0.8	0.7	0.6	0.7
1.0	0.9	0.9	0.8
0.9	0.8	0.8	0.7
0.8	0.7	0.6	0.7
0.9	0.8	0.8	0.7
1.0	0.9	0.9	0.8

0.6	0.7	0.4	0.4	0.4
0.6	0.5	0.3	0.3	0.2
0.8	0.9	0.5	0.5	0.5
0.6	0.7	0.4	0.4	0.4
0.6	0.5	0.3	0.3	0.2
0.6	0.7	0.4	0.4	0.4
0.8	0.9	0.5	0.5	0.5

P++ *Arrays* are distributed over many processors on a parallel machine



P++ *Arrays* are distributed over many processors on a parallel machine

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

Associate an array with geometrical information to get a *MappedGridFunction*

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

Associate an array with geometrical information to get a *MappedGridFunction*

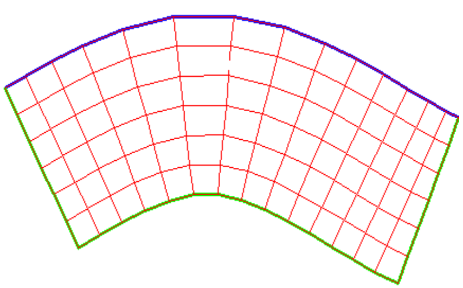
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

Associate an array with geometrical information to get a *MappedGridFunction*

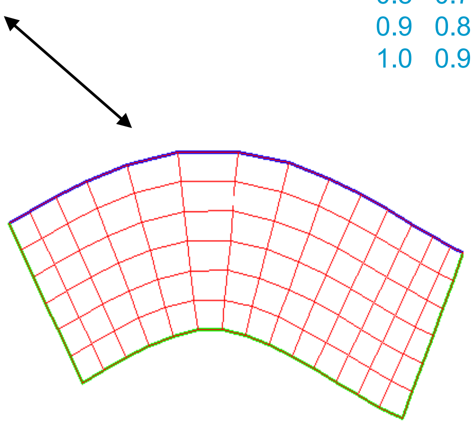
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

A floatMappedGridFunction is *derived* from a floatArray

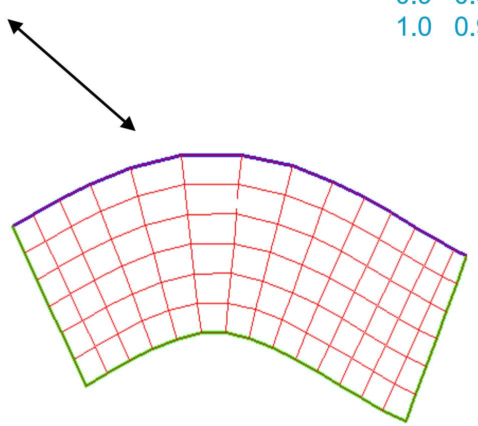


A set of *MappedGridFunction's* forms a *GridCollectionFunction*

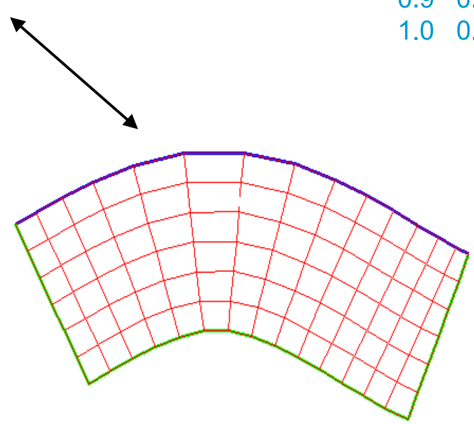
0.8	0.8	0.9	0.5	0.5	0.5
0.7	0.6	0.7	0.4	0.4	0.4
0.7	0.6	0.5	0.3	0.3	0.2
0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.8	0.9	0.5	0.5	0.5
0.7	0.6	0.5	0.3	0.3	0.2
0.7	0.6	0.7	0.4	0.4	0.4
0.7	0.6	0.5	0.3	0.3	0.2
0.8	0.8	0.9	0.5	0.5	0.5
0.7	0.6	0.7	0.4	0.4	0.4
0.7	0.6	0.5	0.3	0.3	0.2
0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.8	0.9	0.5	0.5	0.5



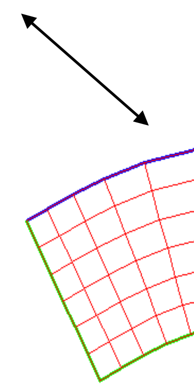
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5



1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

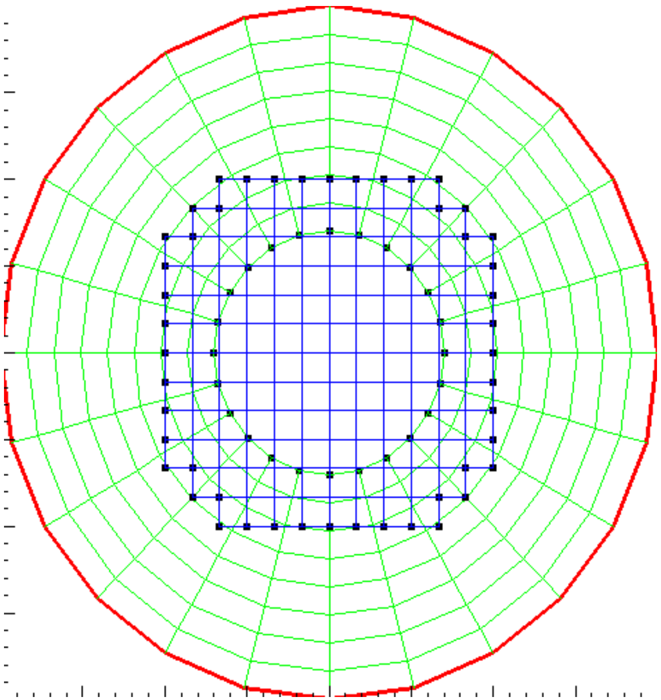


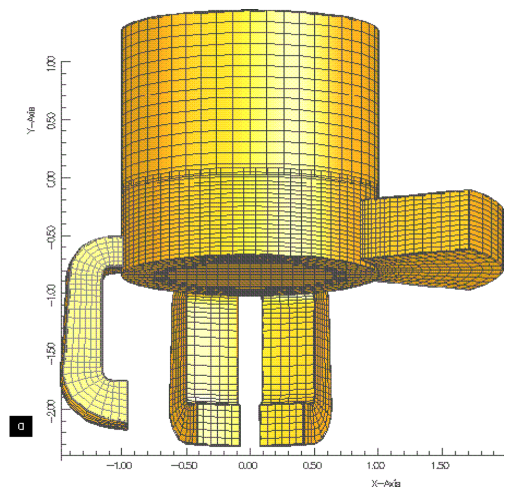
1.0	0.9	0.9	0.8	0.8	0.9	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3
0.9	0.8	0.8	0.7	0.6	0.7	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3
0.9	0.8	0.8	0.7	0.6	0.7	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3
1.0	0.9	0.9	0.8	0.8	0.9	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3
0.9	0.8	0.8	0.7	0.6	0.7	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5

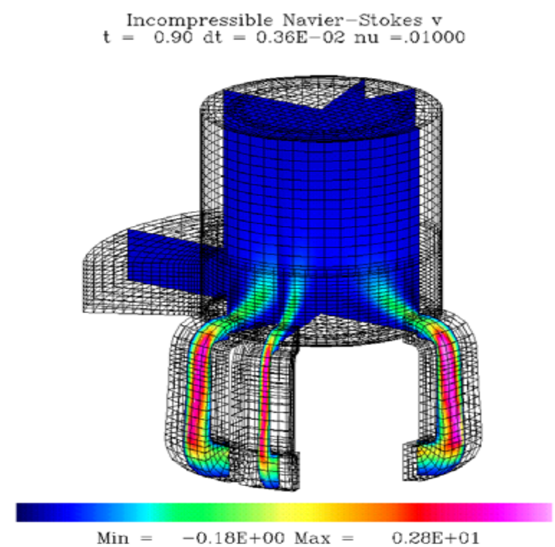


Add information about how grids overlap to get a *CompositeGridFunction*

A CompositeGridFunction is derived from a GridCollectionFunction







In the *Overture* Framework, complex objects behave like built-in types...

```
int i, j,k;
float a, b[10];

j  = i + 10;
b[i] = b[i+1];
```

```
Index I,J;
CompositeGrid cg;
floatCompositeGridFunction u,v,w;
int grid;
w = u + v;
u[grid](I,J) = w[grid](I+1,J-1);
```

In the *Overture* Framework, complex objects behave like built-in types...

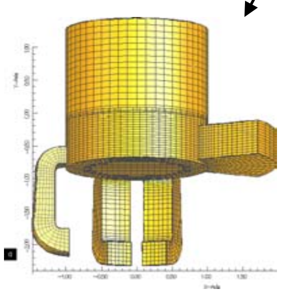
```
int i, j,k;
float a, b[10];

j  = i + 10;
b[i] = b[i+1];
```

```
Index I,J;
CompositeGrid cg;
floatCompositeGridFunction u,v,w;
int grid;
w = u + v;
u[grid](I,J) = w[grid](I+1,J-1);
```

In the *Overture* Framework, complex objects behave like built-in types...

```
int i, j,k;
float a, b[10];

j  = i + 10;
b[i] = b[i+1];
```

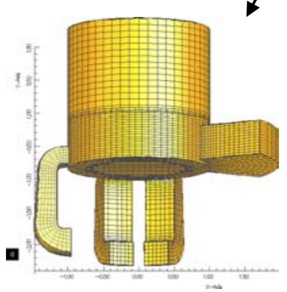
```
Index I,J;
CompositeGrid cg;
floatCompositeGridFunction u,v,w;
int grid;
w = u + v;
u[grid](I,J) = w[grid](I+1,J-1);
```

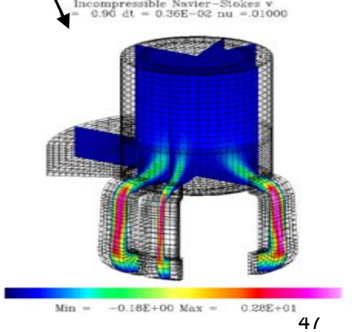


In the *Overture* Framework, complex objects behave like built-in types...

```
int i, j,k;
float a, b[10];

j  = i + 10;
b[i] = b[i+1];
```

```
Index I,J;
CompositeGrid cg;
floatCompositeGridFunction u,v,w;
int grid;
w = u + v;
u[grid](I,J) = w[grid](I+1,J-1);
```

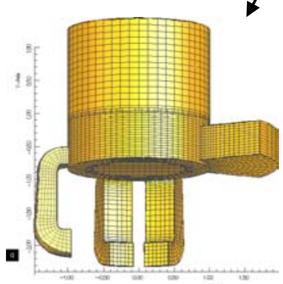


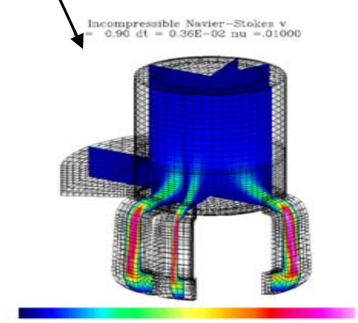


In the *Overture* Framework, complex objects behave like built-in types...

```
int i, j,k;
float a, b[10];

j  = i + 10;
b[i] = b[i+1];
```

```
Index I,J;
CompositeGrid cg;
floatCompositeGridFunction u,v,w;
int grid;
w = u + v;
u[grid](I,J) = w[grid](I+1,J-1);
```



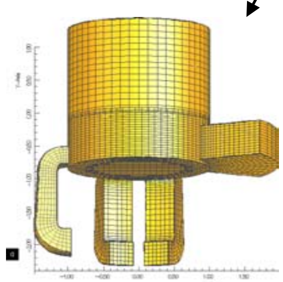


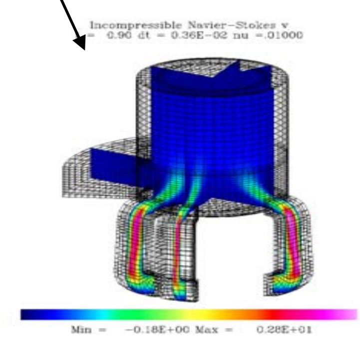
CASC

In the *Overture* Framework, complex objects behave like built-in types...

```
int i, j,k;
float a, b[10];

j  = i + 10;
b[i] = b[i+1];
```

```
Index I,J;
CompositeGrid cg;
floatCompositeGridFunction u,v,w;
int grid;
w = u + v;
u[grid](I,J) = w[grid](I+1,J-1);
```





But in addition, more complex operations are defined for these objects...

```
Index I,J;

CompositeGrid cg;

floatCompositeGridFunction
   u,v,w;


w = u + v;

u[k](I,J) = w[k](I+1,J-1);
```

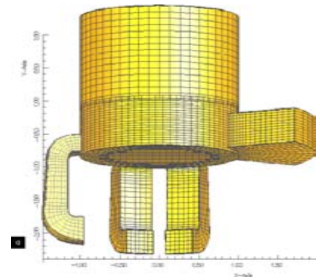
```
w = u.x();    // w = ux
v = u.y();    // v = uy
w = u.xx() + u.yy();
w = u.laplacian();
v = u.div();
```

The differential operators are optimized using Fortran at lower levels

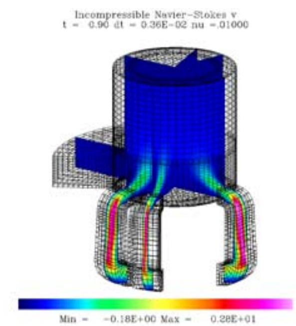
All data and differential operators are also available at lower levels of the data hierarchy, e.g. mesh-point

… and complex operations can be expressed with concise syntax

HDF database access



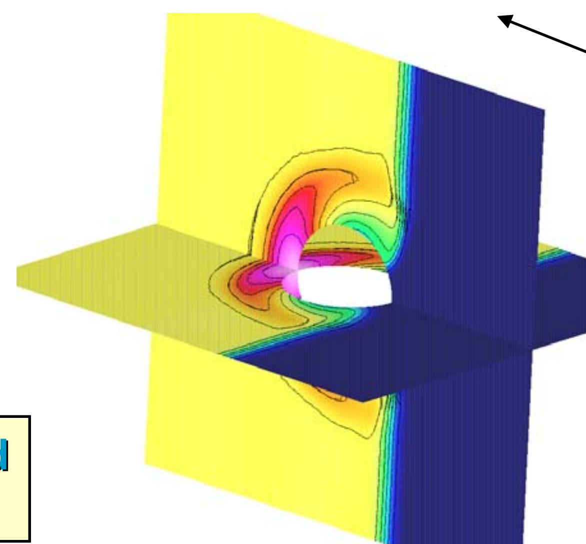
```
CompositeGrid grid;
readFromDatabaseFile (grid, filename);
```




```
floatCompositeGridFunction w(grid, cellCentered, 2);
```

Make a gridFunction on a grid

```
PlotStuff ps;
ps.plot (grid);
ps.contour (w);
```



Visualize grid and data

At the highest level, *Overture* code looks like the underlying mathematics.

Mathematical expressions involving differential operators such as

$$u_{new} = u - \delta t \; ((u \bullet \nabla) u - \nu \Delta u)$$

are expressed concisely using the Overture operator classes.

```
uNew = u - dt*( u.convectiveDerivative() - nu*u.laplacian());
```

This example advances a convection-diffusion equation on an overlapping grid. All grid-dependent and parallel details are hidden at this level.

3D Explicit Convection-Diffusion

```
  float dt=.0005, viscosity=.05;
  for (int step=0; step < 100; step++)
{
    u += dt*( -a*u.x() - b*u.y() + viscosity*u.laplacian());
    u.applyBoundaryCondition (allVelocityComponents, dirichlet, wall);
    u.interpolate();
    if (step % 10 == 0) ps.contour (u);
 }
```

Advance by forward Euler method

Overture interface to library of elementary boundary conditions

Interpolation communicates information between component grids

3D Implicit Convection-Diffusion

Interface to PETSc, Yale, Harwell, Multigrid

Sparse matrix storage of implicit coefficients

```
CompositeGrid grid;
CompositeGridOperators op(grid);
floatCompositeGridFunction coeff(grid);
Oges solver(cg);
```

Derivatives are stored in sparse matrix format

```
for (int step=0; step < 100; step++)
{                                    // ... backward Euler
    coeff = op.identityCoefficients() + dt*(
            a*op.xCoefficients() + b*op.yCoefficients()
            -viscosity*op.laplacianCoefficients() );

    u.applyBoundaryConditionCoefficients
            (allVelocityComponents, dirichlet, wall);

    solver.setCoefficientArray(coeff);
    solver.solve(u, u);

 }
```

Elliptic solver called here; interpolation is automatic

3D Incompressible Navier-Stokes

```
float dt=.0005, viscosity=.05;

for (int step=0; step < 100; step++)
  {                           // ... forward Euler
    u += dt*( -u.convectiveDerivative() + viscosity*u.laplacian());

    u.applyBoundaryCondition (allVelocityComponents, dirichlet, wall);
    u.interpolate();
                              // ... correct by enforcing incompressibility constraint
    u = projection.project (u);
                               // ... visualize
    if (step % 10 == 0) ps.streamLines (u);
  }
```

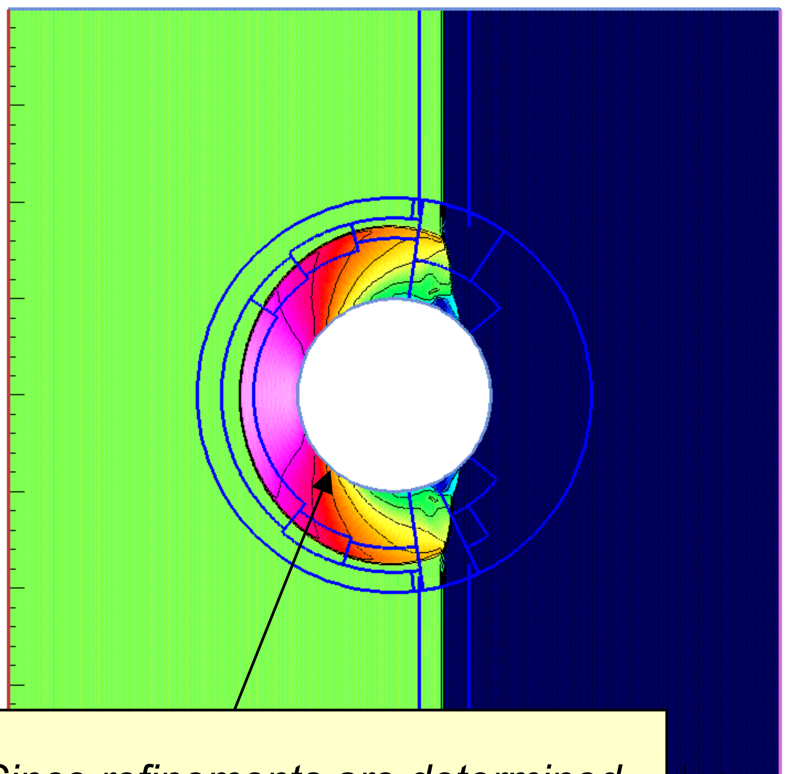
3D Incompressible Navier-Stokes

```
float dt=.0005, viscosity=.05;

for (int step=0; step < 100; step++)
  {                            // ... forward Euler
    u += dt*( -u.convectiveDerivative() + viscosity*u.laplacian());

    u.applyBoundaryCondition (allVelocityComponents, dirichlet, wall);
    u.interpolate();
                               // ... correct by enforcing incompressibility constraint
    u = projection.project (u);
                                // ... visualize
    if (step % 10 == 0) ps.streamLines (u);
  }
```

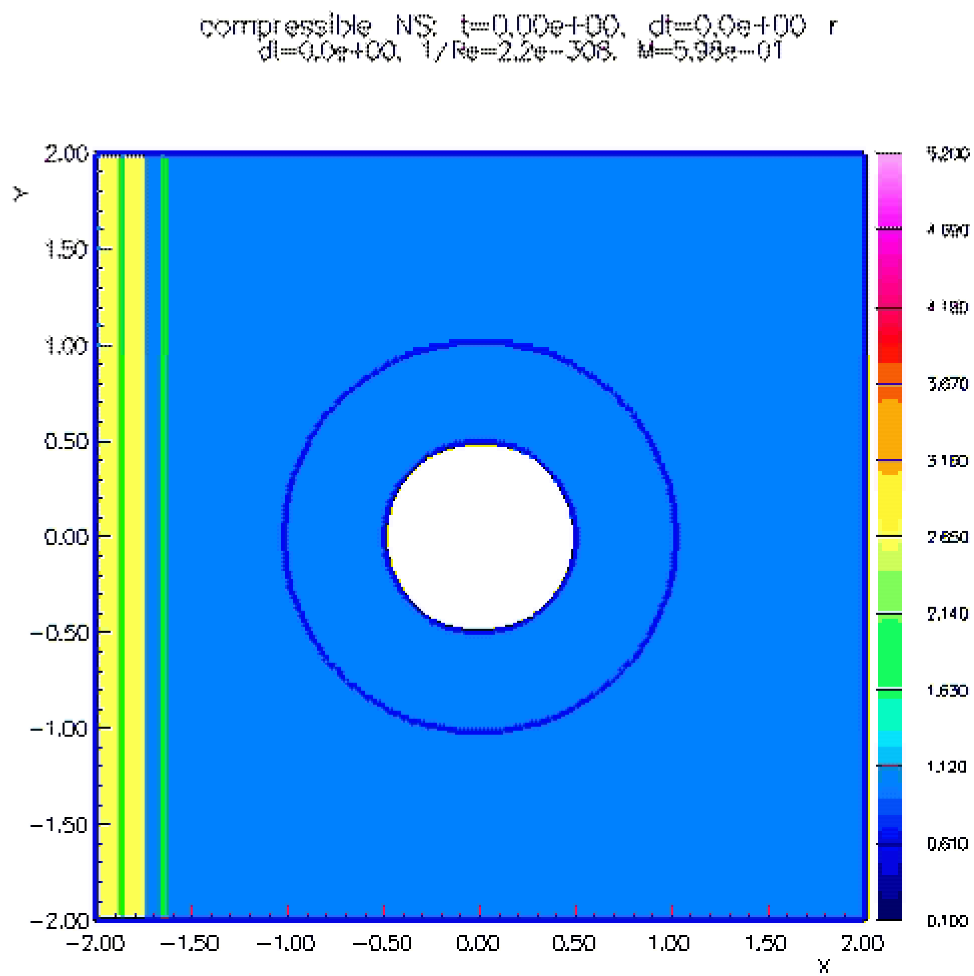
Overture PDE solvers and visualization

- OverBlown Flow Solver
  - Incompressible Flow (pressure-Poisson formulation)
  - Slightly Compressible flow
  - Compressible Flow (Jameson and Godunov methods)
  - AMR
- Overture Visualization Tools

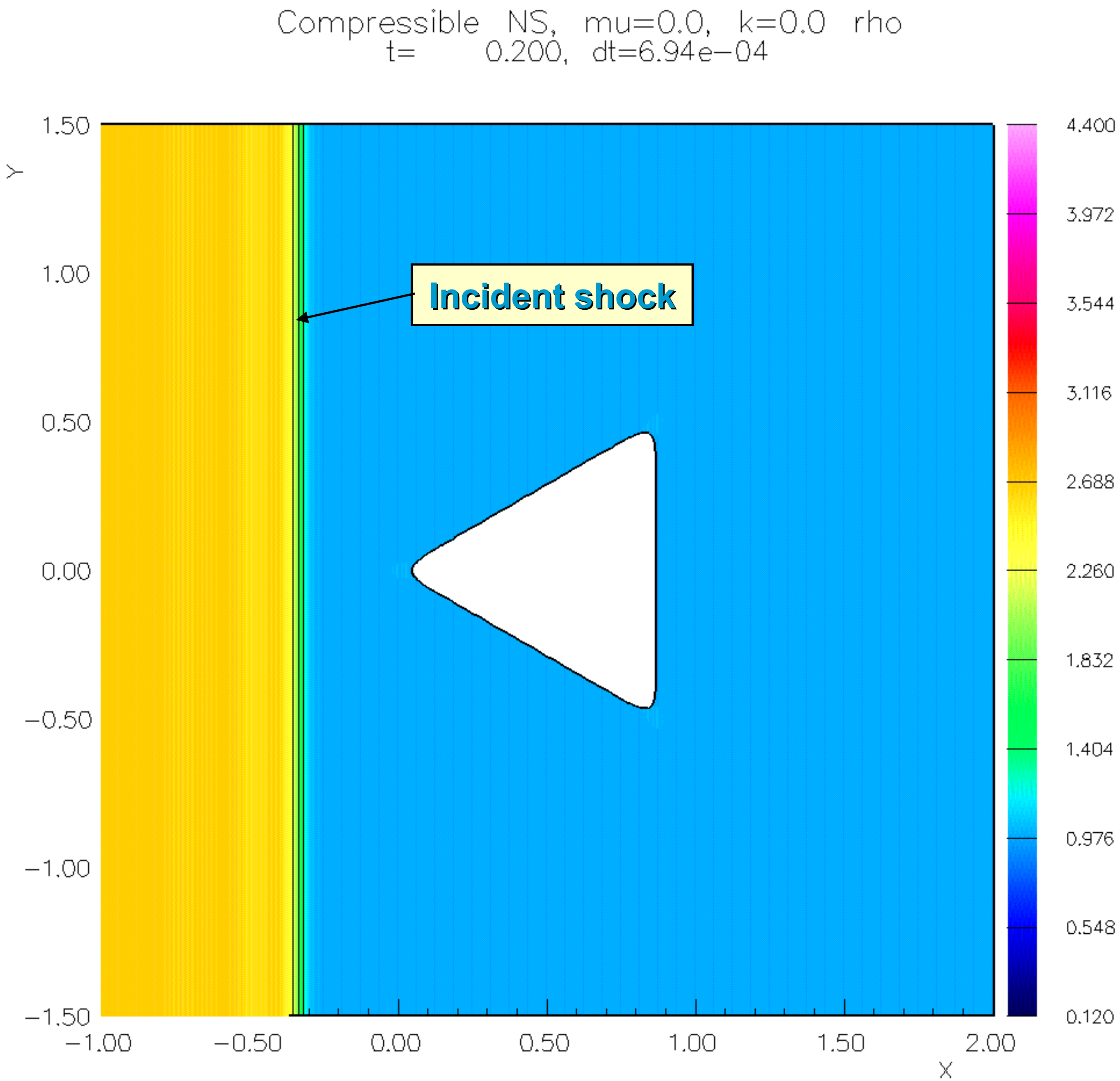
A shock-cylinder interaction is simulated using the *OverBlown* flow solver



Since refinements are determined dynamically, access to original geometry required at run-time



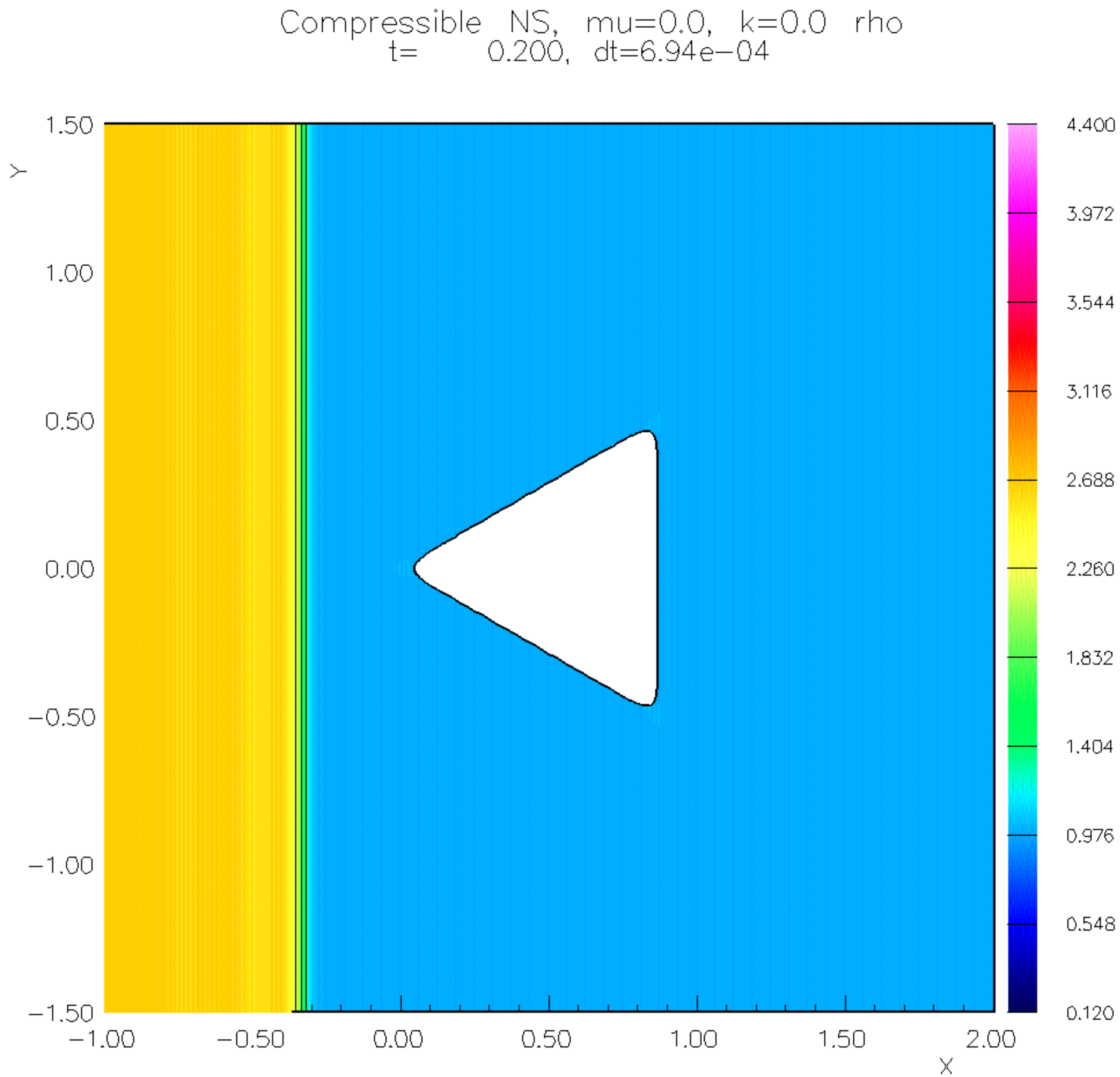
Compressible flow past rounded triangle



Bill Henshaw

CASC

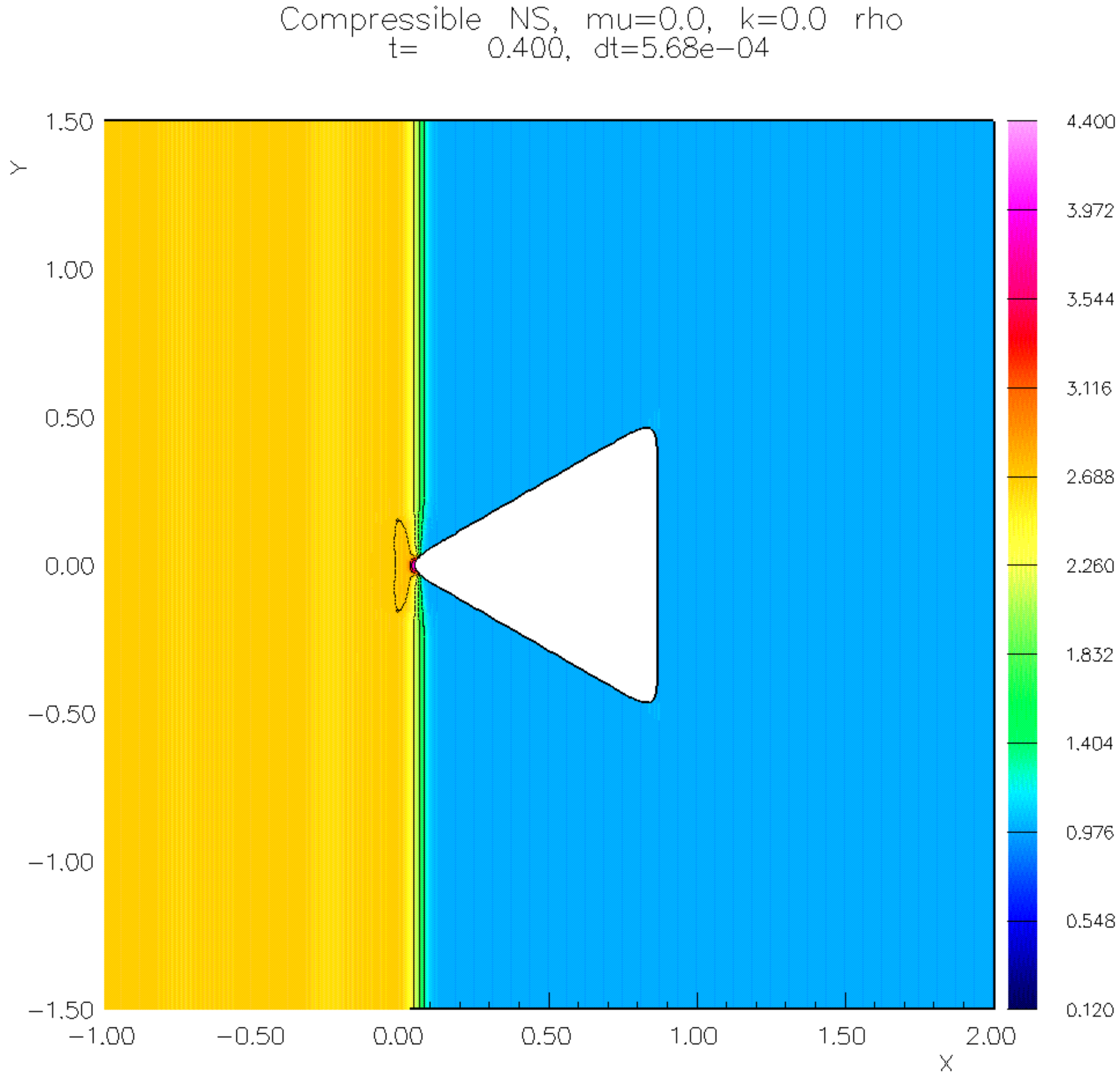
Compressible flow past rounded triangle



Bill Henshaw

CASC

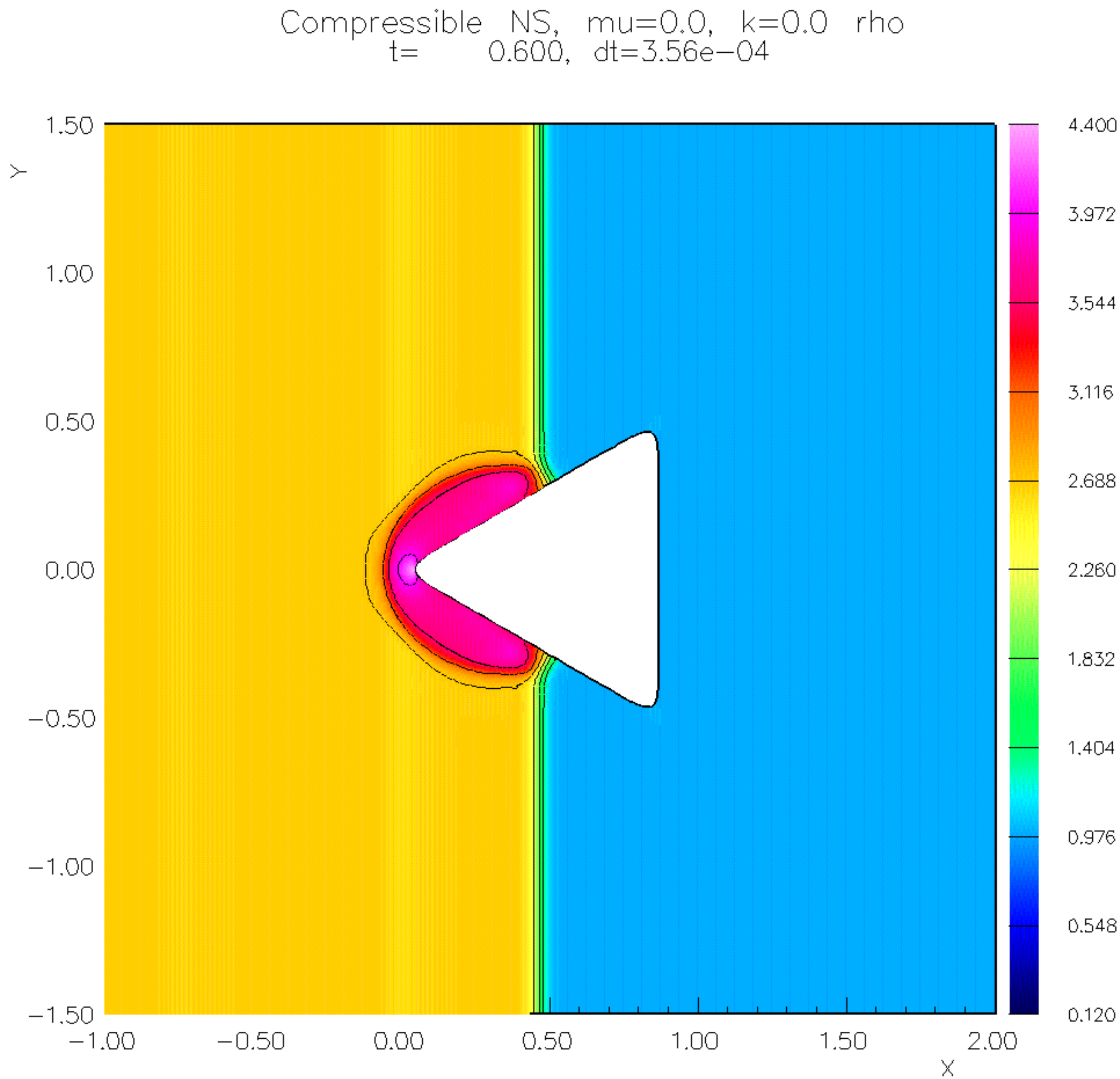
Compressible flow past rounded triangle



Bill Henshaw

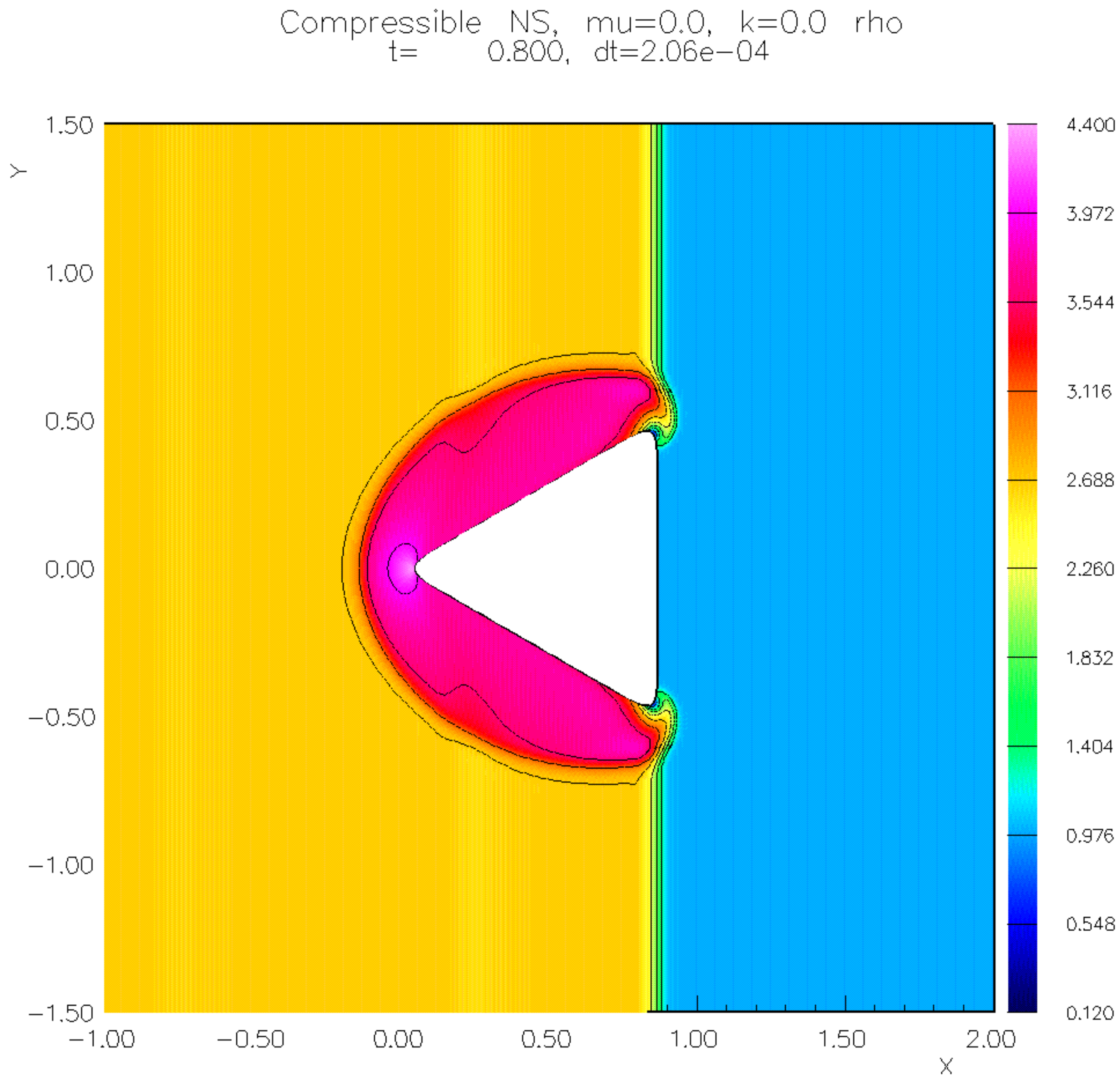
CASC

Compressible flow past rounded triangle



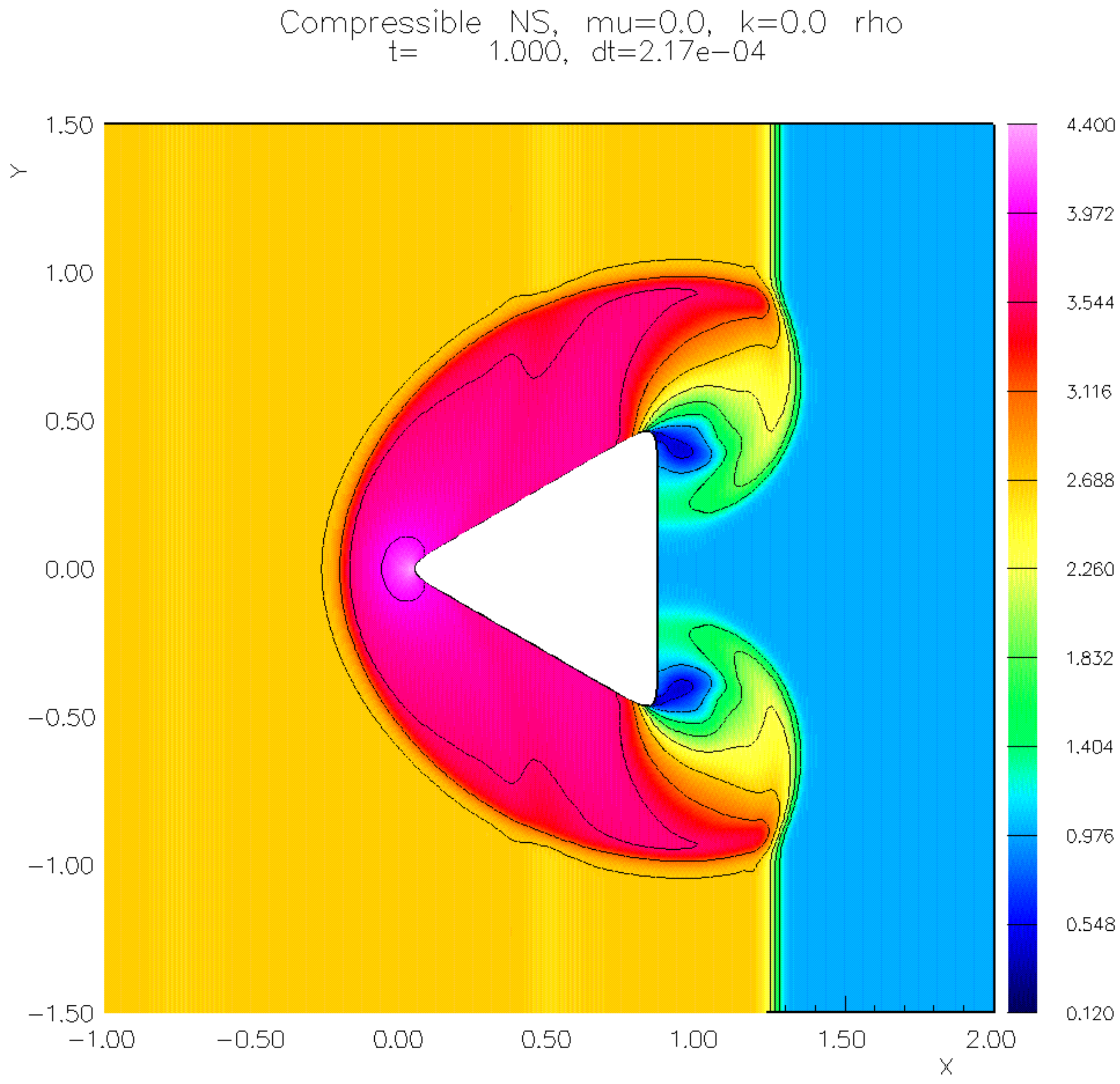
Bill Henshaw

Compressible flow past rounded triangle



Bill Henshaw

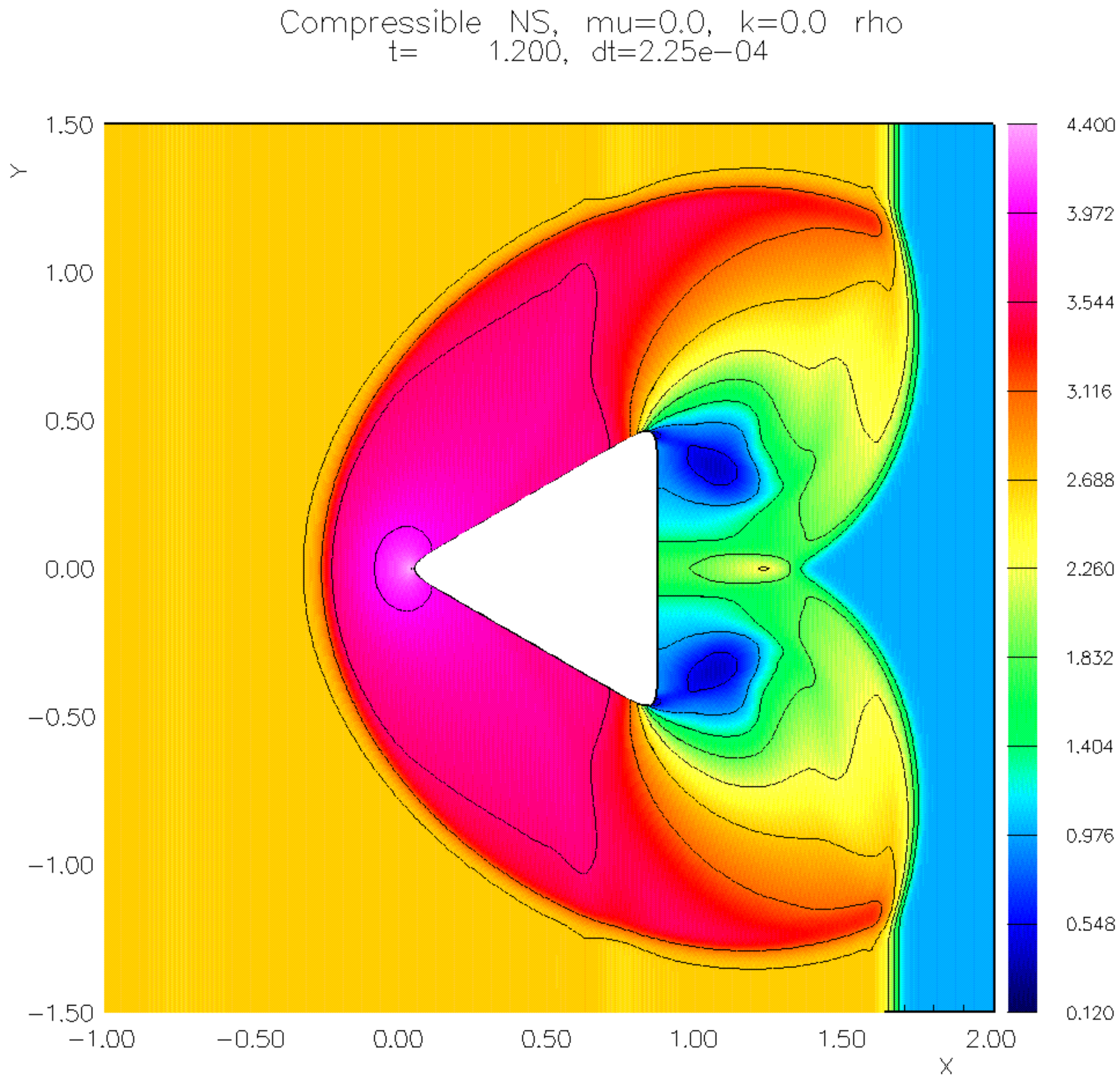
Compressible flow past rounded triangle



Bill Henshaw

Compressible flow past rounded triangle

Bill Henshaw



Obtaining Overture software

- Download from http://www.llnl.gov/casc/Overture
- Full documentation online and in distribution
- Supported architectures:
  - PC-based linux
  - Sun Solaris
  - Compaq
  - IBM*
    - P++ on IBM SP's; most of Overture has not been tested extensively on IBM

UCRL-PRES-150012

Overture Software for Solving PDEs in Complex Geometry

Brown, D.L.

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.